# An XML-based Tool for Verification of Multi-vendor MPLS Router Configuration

by

Hui Wang

B.Sc., Wuhan University, P.R.China, 1998

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

**Master of Science**

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

**The University of British Columbia**

August 2002

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date Sep. 06, 2002

# Abstract

The explosive development of the Internet over the last few years has encouraged multi-vendor networks to support various needs from customers. Multi-vendor configuration management, which is partly concerned with verification of configuration files, is complicated by the fact that each vendor uses a different configuration language.

We have implemented an XML-based configuration verification tool (XVT) for multi-vendor networks. XVT currently can parse Multiple Protocol Label Switching (MPLS) configuration files from Cisco and Juniper companies. XVT can also verify those configuration files against the predefined rules. XVT provides a small set of predefined operators for users to define new verification rules. XML is used in XVT to integrate various router configuration files into one uniform global view, which is used for verification purpose.

# Contents

**Abstract** ii

**Contents** iii

**List of Tables** vi

**List of Figures** vii

**Acknowledgements** viii

**Dedication** ix

**1 Introduction** 1

1.1 Motivation . . . . . 1

1.2 Problems . . . . . 4

1.3 Overview . . . . . 6

**2 Background** 7

2.1 Router Configuration . . . . . 7

2.1.1 Router Configuration Files . . . . . 7

2.2 MPLS VPN . . . . . 11

2.2.1 MPLS . . . . . 11

2.2.2 MPLS VPN . . . . . 12

2.3 Network Management . . . . . 14

2.3.1 Integrated Network Management System . . . . . 16
2.4 XML and Related Technology . . . . . 18
2.4.1 XML . . . . . 18
2.4.2 Related Techniques . . . . . 21
2.5 Related Work . . . . . 22
2.5.1 AT&T-NetDB . . . . . 22
2.5.2 OPNet-NetDoctor . . . . . 22
2.5.3 OpenNMS-BlueBird . . . . . 23
2.5.4 Synopsis . . . . . 24

**3 Design** **25**
3.1 Parse and Check . . . . . 27
3.1.1 Analysis of Configuration Files . . . . . 27
3.1.2 Error Checking Rules . . . . . 34
3.2 Multi-vendor Network Management . . . . . 35
3.2.1 Global View . . . . . 35
3.2.2 Keyword Set . . . . . 40
3.2.3 Query . . . . . 45

**4 Implementation** **46**
4.1 Parser . . . . . 48
4.2 Translator . . . . . 50
4.3 Checker . . . . . 52
4.3.1 Within the Section of the Same Router Configuration File . . 53
4.3.2 Across Different Sections within the Same Router Configuration File . . . . . 55
4.3.3 Across Different Router Configuration Files . . . . . 56
4.3.4 Implementation of Rules . . . . . 56
4.4 Topology Extractor . . . . . 57

4.5 Explorer . . . . . 59
4.6 Query . . . . . 59
4.6.1 The Query Language . . . . . 59
4.6.2 Rules and Query Language . . . . . 65
4.7 Tools for Implementation . . . . . 65
4.7.1 XML Document Parser . . . . . 65
4.7.2 XSLT Engine . . . . . 65
4.7.3 XQL Engine . . . . . 65
4.7.4 Visualization . . . . . 66

**5 Test** **68**
5.1 Simulation of Environment . . . . . 68
5.1.1 Network Topology . . . . . 68
5.1.2 Router Configuration Files . . . . . 70
5.2 Error Report . . . . . 70
5.3 Results . . . . . 70

**6 Conclusion** **73**
6.1 Conclusion . . . . . 73
6.2 Future work . . . . . 74

**Bibliography** **76**

# List of Tables


2.1 Sample of Cisco Router Configuration File . . . . . 9
2.2 Sample of Juniper Router Configuration File . . . . . 10
2.3 Example of XML . . . . . 20

3.1 Juniper's Configuration File Example . . . . . 43

4.1 Parsing Global Setting Variables for Juniper Configuration Files . . 49
4.2 Parsing Global Setting Variables for Cisco Configuration Files . . . . 50
4.3 Mapping for Global Setting Variables . . . . . 50
4.4 Mapping for Interface Variables . . . . . 51
4.5 Mapping for MPLS Label Switched Path Variables . . . . . 51
4.6 Mapping for MPLS VPN Variables . . . . . 52
4.7 Mapping for BGP Variables . . . . . 52
4.8 Predefined Operators . . . . . 62
4.9 Error Rules Expressed by Operators . . . . . 67

# List of Figures


2.1 Network Components of VPN Architecture . . . . . 13
2.2 INMS Model . . . . . 17

3.1 Cisco and Juniper's MPLS VPN Configuration File Sample . . . . . 30
3.2 Cisco and Juniper's VPN BGP Session Configuration File Samples . 31
3.3 Cisco and Juniper's Explicit Path Configuration File Samples . . . 32
3.4 Cisco and Juniper's LSP Configuration File Samples . . . . . 33
3.5 Global View . . . . . 39
3.6 DTD of Keyword Set . . . . . 41
3.7 The Structure of the Keyword Set . . . . . 42
3.8 Parsing and Mapping Procedure . . . . . 44

4.1 Modules of XVT . . . . . 47
4.2 Example of Keyword Set . . . . . 48
4.3 Example of Rules Defined in XSLT . . . . . 57
4.4 Topology . . . . . 58
4.5 Error Report Example . . . . . 60
4.6 EBNF for Query Language . . . . . 61

5.1 Network Topology . . . . . 69
5.2 Error Report Example . . . . . 71
5.3 Performance Chart . . . . . 72

# Acknowledgements

I would like to express my gratitude to my supervisor, Professor Alan Wagner, for his extensive guidence and encouragement. I would also like to thank my parents for their constant love and for their believing me eventual success even when I despaired. Without them, this thesis would never have been completed.

HUI WANG

*The University of British Columbia*
*August 2002*

To my parents

# Chapter 1

# Introduction

## 1.1 Motivation

The explosive growth of the Internet over the last few years has led to the development of diverse equipment provided by different network vendors. Network service provider companies often rely on the network equipment from various vendors to support the customers' needs. However, configuring a large multi-vendor network correctly is extremely difficult for the following reasons[1]:

- *Consistency of neighboring routers*

  The correct operation of the network depends on a consistent configuration across the routers in the network. The operation of an individual router does not ensure the network works properly. For example, two routers need to be correctly configured as each other's neighbor to establish a point-to-point link. The link fails to be established if one of these two routers incorrectly configures its neighbor or if different settings of their configuration do not allow them to inter-operate.

- *Network-wide influence of local changes*

  Changes in the configuration of a single router may have a network-wide influence. For example, for a network in which bandwidth is used as one of the

factors to decide routing paths, increasing the bandwidth of one specific link results in more data moving through this link; thus affecting the traffic across other links.

- *Complex configuration options*

  Router vendor companies provide a wide array of options and parameters for configuration purposes. For example, Cisco's Internet Operating System (IOS) has over 600 commands[1]. What is more complicated is that the router vendors keep updating their products and configuration manuals as their products change. Different versions of the same router product may be different as well. For example, Juniper's JUNOS release 5.3 has new features for improving network security and enabling the Traffic Engineering Database to more readily track and re-act to changes in bandwidth availability, etc..

- *Limited configuration tools*

  Basic commercial tools provide templates for certain configuration operations, such as higher-level languages supporting specific inter-domain routing policies[2], however, the number of commercial configuration tools for large multi-vendor networks is fairly limited. Furthermore, commercial configuration tools often lag behind the release of new routers and techniques.

- *Manual intervention*

  In theory, router configuration data can be automatically driven from a database containing all necessary configuration information. However, in practice complete automation of router configuration is extremely difficult. Manual intervention is necessary when installing new network equipment, fixing network failures and adapting to new demands from users. Manual intervention may produce errors and inconsistencies with unforeseen consequences to the network.

- *Individual standards*

  The network devices from different companies have their own configuration standards. Each vendor-specific product uniquely categorizes and has its own terminology to describe the network components, features, functions and states, even though the underlying concepts and principles are the same for similar products from different vendors.

To minimize the risks of configuration errors, configuration management, which is one of the functional areas in integrated network management[3], is concerned with verification of configuration files. There are a few commercial configuration verification tools at present to help network operators to find and locate the configuration errors in a multi-vendor network. But the number of the tools is very limited. Among these multi-vendor verification tools, the most successful and well-known product is OPNet's NetDoctor[4]. However, users are required to know the Python script programming in order to update the verification rule library. For other multi-vendor network tools, Lucent's OneVision Management System[5] is only enabled for ATM, Frame relay and IP networks, while Dorado's RedcellTM Virtual Private Network Service Center (VPN SC) system[6] automates the creation and deployment of IP VPNs but ignores configuration verification.

This thesis introduces the design and implementation of a flexible, extensible configuration verification tool for routers in multi-vendor networks. The contributions of the thesis include the following:

- We develop a XML-based router configuration verification tool, XVT, for multi-vendor networks. Given the router configuration files from different vendors, the tool checks the configuration errors in the individual file as well as across these files. The tool is extensible in the sense that users can add new verification rules using predefined verification operators.

- We explore the advantages of XML used in the configuration management of

multi-vendor network.

## 1.2 Problems

The scope of the problems discussed in the thesis is limited in the following ways:

- Network architecture

  The architecture of the network considered in this thesis is MPLS VPN architecture. The MPLS VPN network is a multi-vendor network with routers from Cisco and Juniper vendor companies. The network protocols configured in this architecture are Multiple Protocol Label Switching (MPLS) and Border Gateway Protocol (BGP).

- Network devices

  There are various network devices such as routers, switches, and bridges working in one multi-vendor network. However, routers are the only network devices that are discussed in the thesis.

- Data from network

  Data in a multi-vendor network can be grouped as static data and dynamic data. The static data is the data that does not change frequently and is comparatively static, for example, the router configuration data saved in the configuration files. The dynamic data is the data that keeps changing from time to time, for example, the traffic information data in the network. The network data considered in this thesis is only static data from router configuration files.

  There are various kinds of router products from different vendors. Cisco and Juniper are two leading companies in the router market and together have more than 90% of the router product market. Hence Cisco and Juniper router

configuration files are chosen for analysis since they represent most router products in the market at present.

- Network management function area

  The International Organization for Standards (ISO) has defined five function areas for network management. These five areas are fault management, accounting management, configuration management, performance management and security management[7]. This thesis only considers the configuration management area. Specifically, the thesis discusses the topic of router configuration verification.

The basic goal of the thesis is to implement a flexible, extensible router configuration verification tool for multiple vendors. Although standardizing configurations has been mentioned in the literature, a common standard has not emerged. Router vendors tend to use private standards to highlight the aspects of their products that differentiate them from their competitors. As a result configuration files from different router vendors are typically quite different from each other. Developing a separate verification tool for routers of each vendor is one approach. However, it cannot exploit the commonality that exists for features and attributes among different configuration languages. Another approach is to have a general configuration verification tool to check against the configuration files of various standards from different vendors.

The tool needs to generalize various configuration information from different routers and collect the generalized configuration information. The representation of the collected data is the fundamental problem for the implementation. How to define the universal set of configuration information as well as the mapping rules between the configuration data and the universal set will be discussed in the thesis. The design of a query system to enable users to create new verification rules and query without specific programming skills is also discussed in the thesis.

XML has been standardized within the WWW Consortium (W3C) with the aim of bringing mark-up to data representation. XML has also been found useful in data processing and system integration due to its ability to be machine accessible but in a human understandable way. It can express a huge range of document types because of its separation of data content from how it is represented. As a result it has the potential to be used in the specification of management information and interfaces. The problem we explore is whether there are advantages and disadvantages of using XML for the specification of information in a multi-vendor router configuration verification tool.

## 1.3 Overview

Chapter 2 introduces the background knowledge and related work for multi-vendor network management as well as the basic concepts of XML. Chapter 3 presents the ideas of the design. How to define the generic data repository and how to map the managed configuration data into the generic data repository is discussed in Chapter 3. The idea of a keyword set, which is for extensibility purpose, and the design of the verification rules and queries are also introduced in Chapter 3. Chapter 4 describes the details of implementation of the tool. Chapter 5 presents the tool's experiment environment and results. And the last chapter presents conclusion and future work.

# Chapter 2

# Background

## 2.1 Router Configuration

The basic function of a router is to receive incoming packets and direct them towards the destination. Routers also participate in various protocols to provide network services such as MPLS or BGP, and higher-level user services such as multicast. In addition, routers may perform various filtering functions to control the packets that will be accepted or denied by the network. These tasks are configured in the router by setting a large number of configuration options in the configuration file that control the operation of the router.

### 2.1.1 Router Configuration Files

In general, router configuration files have the following features:

- Various keywords

  Different vendors use their own configuration language to configure their routers. Different configuration keywords are used in different router configuration files even for the same configuration task. For example, in a Cisco configuration file the keyword of a label switched path is "tunnel" while in Juniper it is "LSP".

- No common file structure and access method

  Router configuration files are text files with a sequence of characters organized in lines. Some configuration files are simply streams of bytes without any structure while other configuration files are formatted rigidly. Some configuration files are accessed sequentially so that each text line in the configuration file is processed in order, one after another. Some other configuration files are accessed directly so that arbitrary lines can be read or written without restrictions on the order. For example, Cisco router configuration files are free form and manipulated with sequential access. Juniper router configuration files are represented in XML format with tree structure and manipulated with direct access.

### Cisco Router Configuration Files

Cisco router configuration files consist of four major components: global settings, interface specification, filter specification, and routing protocol specification[8]. The "global settings" component consists of basic configuration parameters such as hostname and password encryption. For example, the router configuration file in Table 2.1 includes two parameters for its "global settings" component: "version 12.0" which identifies the version of router product and "hostname Pescara" which specifies the router name. In addition to the global settings, the other three components configure different aspects of the router configuration. The "interface specification" component performs the configuration of interfaces such as setting the IP address of interface. The "filter specification" component performs the configuration of packet filters and route filters. Packet filters are configured via an access list that identifies which packets should be accepted or denied. Route filters are configured via a road map that identifies the routing between different Autonomous Systems (ASs). The "routing protocol specification" component performs the configuration of routing protocols.

```
!
version 12.0
!
hostname Pescara
!
interface Loopback0
  ip address 10.10.10.4 255.255.255.255
  ip router isis
!
interface Loopback101
  ip address 200.0.4.1 255.255.255.0
  no ip directed-broadcast
!
router bgp 100
   bgp log-neighbor-changes
   neighbor 10.10.10.6 remote-as 100
   neighbor 10.10.10.6 update-source Loopback0
!
```

Table 2.1: Sample of Cisco Router Configuration File

The configuration commands can be grouped into different sections corresponding to the components they belong to. For example, the router configuration commands in Table 2.1 can be grouped into three sections: "global settings" section for the "global settings" component, "interface" section for the "interface specification" component and "router BGP" section for the "routing protocol specification" component. The "global settings" section includes two parameters, "version" and "hostname". The "interface" section includes the configuration for two interfaces: Loopback0 and Loopback101. The "router BGP" section consists of BGP routing session configuration with the neighbors in the BGP peer group identified in the "router BGP" section.

There are two types of commands in Cisco configuration files. Each type of command has different formats. The first type of the command is to assign values to the variables. It follows the "variable value" format. The variable is also called a "keyword". For example, "hostname p1" is the command used to configure the

router name. The literal "hostname" is the keyword of the command while "p1" is the data value. The second type of command is to disable/enable condition variables. It follows the "keyword" format. For example, "tag-switching ip" command is used to enable MPLS protocol on one specific interface with the "tag-switching ip" as the keyword.

### Juniper Router Configuration Files

Juniper router configuration files are represented in XML format with a clear hierarchical relationship[9]. For example, the Juniper router configuration file shown in Table 2.2 configures MPLS and LDP protocols on the router. The MPLS protocol is enabled on interface "s0-0/0/0/0" and LDP protocol is enabled on interface "s0-0/0/1/0". The "protocols" tag is the element with two sub-elements; each protocol is one sub-element.

```
<protocol>
  <mpls>
    <interface>
      <name>s0-0/0/0/0</name>
    </interface>
  </mpls>
  <ldp>
    <interface>
      <name>s0-0/0/1/0</name>
    </interface>
  </ldp>
</protocol>
```

Table 2.2: Sample of Juniper Router Configuration File

Since Juniper router configuration files are represented in XML format, the commands in Juniper router configuration files are represented in matched-tag format. These tags are context-sensitive, i.e., the meaning of one specific tag depends on its ancestor elements, its sibling elements and its children elements. One configuration command is represented by a set of tags. For example, the command to

configure the interface for MPLS protocol name in Table 2.2 is represented by a set of tags with "mpls" tag as the root and "name" tag under the "interface" tag. The "name" tag itself cannot specify the interface for MPLS without any context from ancestor elements.

## 2.2 MPLS VPN

### 2.2.1 MPLS

Multiple Protocol Label Switching (MPLS) is a high-performance packet forwarding technology that integrates the network layer (layer-3) routing and traffic management capabilities with the data link layer (layer-2). In MPLS, packet forwarding is done based on short, fixed length labels. A label switched path (LSP) defines a sequence of labels from the source to destination so that each MPLS router is able to forward the packets following the LSP.

The routers participated in MPLS are classified into Label Switching Routers (LSRs) and Label Edge Routers (LERs). A LER is the router located at the edge of MPLS network and the access network. It is used to establish the Label Switched Paths (LSPs), and assign or remove the packet labels when traffic enters or leaves the MPLS network. The router that assigns the initial label for traffic entering the network is called the ingress router, while the router that removes the labels for traffic leaving the network is called the egress router. A LSR is a router that is not located at the edge of MPLS network and used to exchange the labels of the packets to enable packets to be forwarded to the next hop. The basic MPLS configuration includes the following commands[9][10]:

- Enable MPLS on the router

  MPLS is enabled on the router by configuring MPLS on one or more specific interfaces of the router. Meanwhile, MPLS protocol is added to the "protocol" section in the configuration file.

- Configure MPLS-signaled LSPs

  MPLS-signaled LSPs are created to enable traffic between the ingress router and the egress router. Only the ingress router is configured to create the LSP; all other routers need not to be configured.

  To configure signaled LSPs, the first step is to create one or more named paths on the ingress router. For each named path, some or all transit routers can be specified in the path, or left empty. The second step is to create one or more LSPs and define the properties associated with the LSP on the ingress router. The name of the LSP comes from the named paths defined in the first step.

### 2.2.2 MPLS VPN

Virtual Private Network (VPN) technology is based on a *tunneling* strategy. Tunneling involves encapsulating packets in a base protocol format with some other protocols. In MPLS VPN architecture, the packets are encapsulated with MPLS packet format for forwarding purposes.

MPLS VPN architecture shown in Figure 2.1 consists of the following network components[11]:

**Customer Edge Devices (CE)** CEs provide customers the access to the service provider networks over a data link to one or more provider edge routers.

**Provider Edge Devices (PE)** PEs exchange the routing information with the connected CEs using static routing. PEs also exchange the routing information with other PEs to learn remote routing information. In MPLS VPN architecture, a PE plays another role as a LER. Hence the PE is responsible for setting up the LSPs, assigning labels and removing labels from packets.

**Provider Devices (P)** P routers are the routers that do not have a direct connection to the CE devices. P routers function as transit LSRs in MPLS VPN architecture to forward the packets.

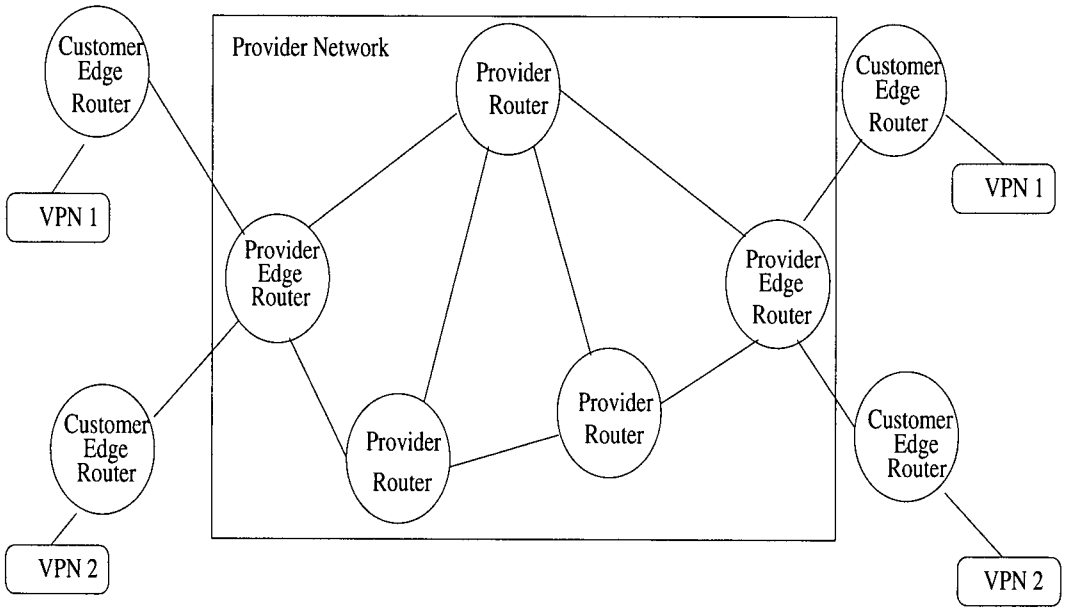



Figure 2.1: Network Components of VPN Architecture

Each VPN is associated with one or more VPN Routing / Forwarding instances (VRFs). A VRF defines the VPN membership of a customer site attached to a PE router. The target VPN community is a set of sites within a VPN to which a PE router distributes routes. The distribution of VPN routing information is controlled through the use of VPN route target communities, implemented by Border Gateway Protocol (BGP) extended communities. The basic MPLS VPN configuration includes [11][12]:

- Define VPNs

  The definition of VPNs includes creating a VRF to be associated with one specific VPN routing instance, creating a list of import and export route target communities for the specified VRF, and associating a VRF with an interface. The configuration of VPNs occurs only on the PE routers.

- Configuring BGP routing sessions between PE routers and CE routers

The BGP session is configured between a PE router and a CE router to set up the connection between them. For each CE router, the PE router keeps one specific VRF. Using the connection, the CE router advertises the site's local VPN routes to the connected PE router and learns remote VPN routes from the same PE router. The traffic from the CE router enters the VPN provider network via PE routers and is distributed in the core network following the specified VPN route target communities.

- Configuring BGP routing sessions between PE routers

  The basic configuration of BGP is to configure the connection between the PE routers. After exchanging the data with CE routers, the PE router continues to exchange the routing information with neighbor PE routers using BGP. The remote routing information is learned from the exchange.

- Configuring MPLS on P routers

  The P routers play the role of the LSRs in MPLS VPN. MPLS must be enabled on all P routers. The configuration of MPLS is the same as that introduced in Section 2.2.1.

- Configuring MPLS LSP tunnel between PE routers

  MPLS LSP tunnel is configured only on the ingress PE router that is the edge router from where the traffic enters the provider network. The configuration of MPLS LSP tunnels is the same as that introduced in Section 2.2.1.

## 2.3 Network Management

Rapid development in computer and network technology has led to an explosion in the variety of equipment and networks offered by vendors. The network service provider companies do not only rely on a single vendor and a relatively straightforward architecture to support customers' needs. They need strategies to manage

their existing equipment and the equipment they may acquire. Consequently network management is an important area whose goal is to help the network operators organize and monitor the network activities and resources.

Network management covers the areas of planning, configuring, controlling, monitoring, fault correction and administration of networks. The purpose of network management is to manage the network and its components to be used at a reasonable cost and with optimum capacity[13].

A network management system consists of at least one management station; several (potentially many) nodes, each with a processing entity, termed an agent, which has access to management instrumentation; and a management protocol, used to convey management information between the agents and management stations[14]. Operations of the protocol are carried out under an administrative framework which defines both authentication and authorization policies.

There are several standards for TCP/IP network management. The Internet Engineering Task Force (IETF) has recommended Simple Network Management Protocol (SNMP)[15] as the standard network management protocol for Local Area Networks (LANs). SNMP is used as the management protocol to communicate values of variables stored in a Management Information Base (MIB)[16] that summarizes the states of the router including basic traffic statistics. There are several versions of SNMP. The first version, SNMPv1, is the original Internet-standard Network Management Framework, is described in RFC1155, RFC1157, and RFC1212. SNMP version 2, SNMPv2, which is described in RFCs 1902-1910, is the SNMPv2 Framework derived from the SNMPv1.

Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the Management Information Base (MIB). Collections of related objects are defined in MIB modules. These modules are written using a subset of OSI's Abstract Syntax Notation One (ASN.1)[17]. The MIBs typically do not provide full details of the configuration of routing protocols and

access lists.

### 2.3.1 Integrated Network Management System

Multi-vendor is becoming a common term for describing today's network environment. In most of the current networks, there are products and services from different vendors. The evolution of a typical network always starts with a small set of homogeneous products, and grows in size and complexity with a wide variety of complex, interconnected technologies that cannot always be known beforehand.

The management of a multi-vendor network is complicated by the fact that each vendor uses a different management system. The problems with having different management systems are that they[3]:

- apply to a very limited and specific set of network components;
- use vendor-specific management data with its own syntax and semantics; and
- employ different interfaces to the network operator.

As a consequence, operators are required to be trained for each vendor-specific management system. This training is needed because of the specific terminology of each vendor-specific product used to describe the network components, features, functions and states even though the underlying concepts and principles are the same for similar products from different vendors.

One approach to address the need to efficiently and effectively manage multi-vendor networks is to provide an Integrated Network Management System (INMS). The INMS model shown in Figure 2.2 is based on the following architectural components [3]:

- Unified user interface.
- Standards on presentation of network management information to users.

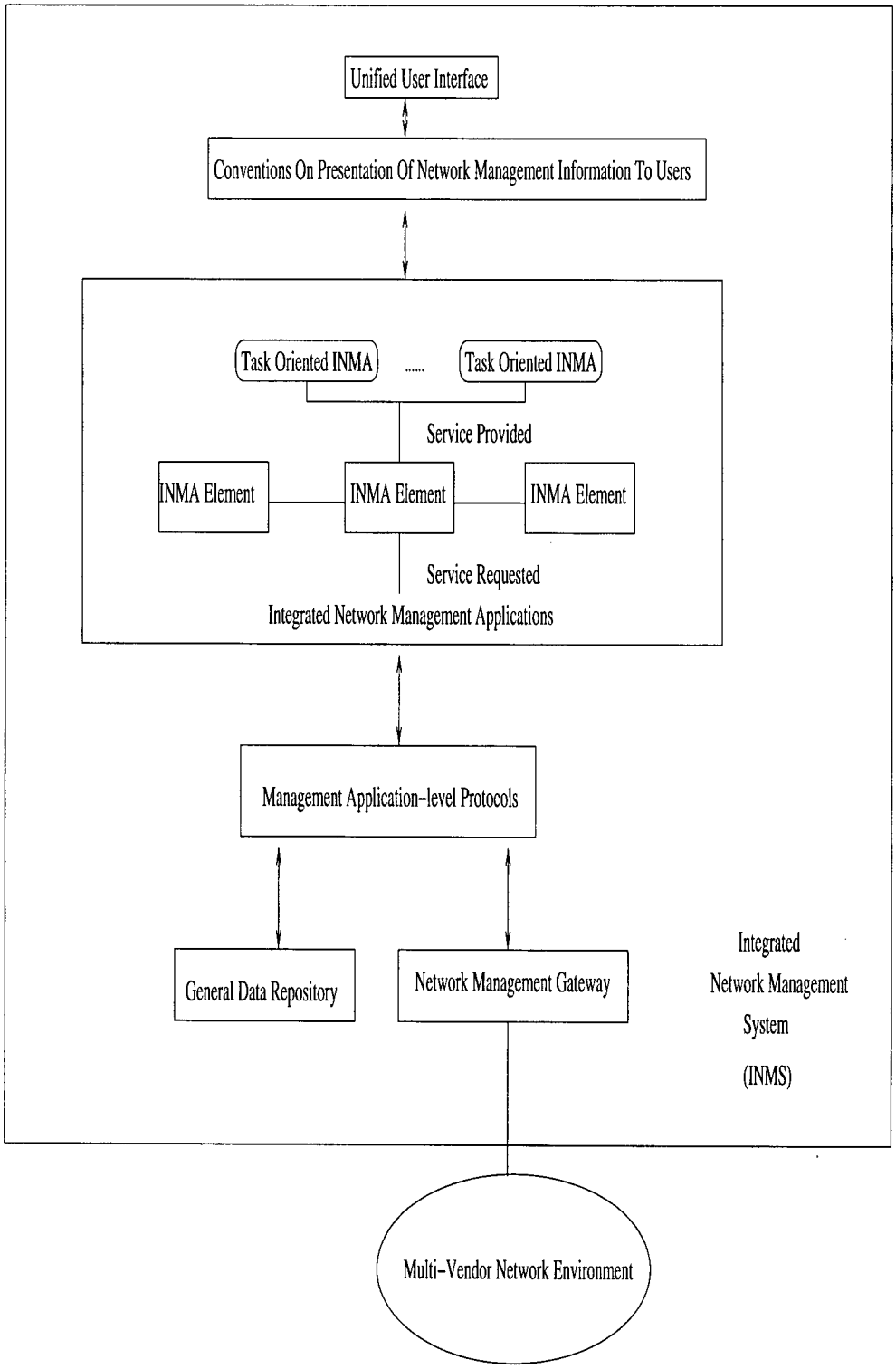



Figure 2.2: INMS Model

- Family of Integrated Network Management application elements.
- Protocols on management application level.
- Management of data repository.
- A set of communication protocols and services.

Among these components, providing a unified and generic data repository of a multi-vendor network is important because the management protocols, user interface and INMS applications are all based on it. The key to the approach is how to define the generic data repository to describe the managed information in the network and how to map the real network information into the generic data repository.

## 2.4 XML and Related Technology

Extensible Markup Language (XML) has emerged as a new standard for data representation and exchange. Many XML related technologies make XML a powerful and useful technology..

### 2.4.1 XML

XML[19] brings a set of markers, called tags that define the function and hierarchical relationships of the parts of a document or data set, to data representation. The tags are similar to Hypertext Markup Language (HTML) tags, but unlike HTML XML is used to define tags that best suit the data being marked.

An XML document is a sequence of elements, each consisting of free text and/or other elements. The element tags must match. For example, as shown in Table 2.3, each <person> must match with </person>. The elements in XML loosely correspond to the objects in an object-oriented language. Nested XML elements represent fields of an object. For example, as shown in Table 2.3, treating

the element "person" as the "person" object, the element "telephone" is nested as the child of the element "person" and represents one of the "person" object's attribute fields. Users can produce XML files with complex structure without first having to define the schema. Furthermore users can define their own tags to describe the objects. The advantages of XML are followings:

- Flexibility

  XML is a semi-structured document language that lies between more structured documents such as relational database tables and completely unstructured free text documents. XML uses tagged paragraphs of free text, where each paragraph contains relevant information for a domain of interest.

  XML is more flexible in capturing data than database forms and structured documents because there is no rigid structure requirement. XML is easier than free text documents in managing information since free text documents require sequential access while XML uses direct access.

  XML is more flexible in defining the tags or schema than HTML files and relational database tables. The tags or schema of XML file are not predetermined. Users can define the name of tags and the relationship of the tags in the way they prefer. Using tags the data content is separated from how it is represented. Hence XML can be used to express a huge range of document types.

  XML is also flexible for its loose tree structure and arbitrary nesting. In a relational database the structure of the datasets is fixed. The data schema defines exactly how many elements one data field has. For example, if the person's telephone number field is defined for only one value, there should be no record with two telephone numbers. If the user wants to store the information for people who have two or more telephone numbers the schema must be changed. But changes to the schema will result in redundancy for

records with only one telephone number.

```
<person name = "David" >
  <telephone number="1234567"/>
<person>

<person name = "Jane" >
  <telephone number="2347856"/>
  <telephone number="1234568"/>
<person>
```

Table 2.3: Example of XML

However, in XML arbitrary nesting is permitted inside the structure. For the same example mentioned above, the problem is solved using the XML file shown in Table 2.3.

- Easy for transformation and query

  XML is not only a data representation language but also a data management tool because of the availability of techniques for querying and transforming XML documents. These techniques include extensible Stylesheet Language Transformation (XSLT) and XML Query Language (XQL). XSLT enables XML files to be easily transformed into HTML, plain text, or XML files with different schemas. XQL enables querying about data represented in XML format.

- Various application tools

  Numerous application tools are implemented for browsing, editing and processing XML documents, offering cost savings in application development[20]. Such tools allow users to build XML documents and update the content of XML documents. For example, the tools based on the Document Object Model (DOM), which is designed to be used with any programming language, enable users to navigate the XML documents' structures and to add, modify

or delete elements and content from the XML documents. Furthermore there are many application tools for other XML-based application categories, for example, XSLT engines provide Application Programming Interfaces (APIs) for users to access an XSLT stylesheet and transform the XML documents into other formats, e.g., an HTML file. Those tools avoid the coding work for developing XML-based applications.

### 2.4.2 Related Techniques

#### XSLT

XSLT[21] is designed as part of eXtensible Stylesheet Language (XSL), which is a stylesheet language for XML. The XSLT file defines the transformation rules. The original XML file can be transformed into another XML file, HTML file or plain text, by applying an XSLT file. If the output is in XML format, the transformed XML file may have the same or different markup and structure compared with the original XML file.

#### XQL

As an increasing amount of information is stored, exchanged, and presented using XML, the ability to intelligently and efficiently query XML data sources becomes important. There are a few XML-based query techniques such as XML-QL[22], XQuery[24], and XQL[25]. XQL is designed to be a language for addressing and filtering the elements and text of XML documents following the queries. The results of XQL queries are also in XML format. There are some good XQL engine tools available, e.g., GMD-IPSI XQL engine from http://xml.darmstadt.gmd.de/xql. But as yet there is no engine tool available for XML-QL and XQuery.

## 2.5 Related Work

There are several commercial configuration verification tools. Some of the tools are distributed with the router products by the specific vendor. Some are distributed for general configuration management use.

### 2.5.1 AT&T-NetDB

NetDB[8] is a network configuration tool developed by AT&T for Cisco routers. It comprises a network-wide view from the configuration files of all Cisco routers for an IP network. Based on the network-wide view users can browse the network structure and search for network components. NetDB also checks the possible router configuration errors and configuration inconsistencies based on the knowledge of Cisco IOS.

NetDB's internal data model[8] is implemented using Perl hash tables. The keywords and their values in the configuration file are stored separately in two hash tables. One table contains the keywords that are used in the original configuration file and the other one contains the value for each keyword. Every time a specific data field is queried, the keyword table is searched first and then the value table is searched following the matched entry. Separating the entry and value increases the processing time thus adding to the inefficiency of the system.

NetDB can only parse and validate Cisco IOS router configuration files[8] and cannot be used in the multi-vendor network that consists of network devices from vendor companies other than Cisco.

### 2.5.2 OPNet-NetDoctor

OPNet is a commercial simulation product developed by MIL3 Company. OPNet provides a troubleshooting and validation tool named NetDoctor for network configuration.

NetDoctor[4] allows users to identify potential or existing trouble spots in the network, as well as determine whether or not the current network configuration is the most optimized. NetDoctor seamlessly interfaces with OPNet's Virtual Network Environment enabling users to preview the effects of various configuration changes before implementing them in the operational network.

NetDoctor can parse the router configuration files from various vendors with different technologies. OPNet's network model enables configuration files from different vendors such as Cisco, Juniper and Nortel, to be imported into the tool. It also allows users to choose the network devices from the model family. Furthermore, NetDoctor provides an embedded rule library for checking the consistency of configurations, against an established set of rules to identify configuration errors or warnings. Users can add new rules to the set of rules when the devices are updated with new features. The rules are implemented programmatically in the Python scripting language where each rule is mapped to one specific Python script. Users must be familiar with the Python script language and NetDoctor's representation of the network to add new verification rules.

### 2.5.3 OpenNMS-BlueBird

The BlueBird project is a so-called "next-generation" network management tool[27] which addresses the potential use of XML and XSL into network management. XML plays the following roles in the BlueBird project[28]:

- Adaptive Configuration

  Rule-based configuration promotes ease of administration and minimizes maintenance using easily understood drag/drop rule metaphors. The rules and all configurations are stored in XML so that any XML editor tool can manipulate them.

- Data Reporting

Report network information that is saved in XML documents following user's choice of formats with XSL transformation capabilities.

- Web presentation

  Usage of Java, XML and XSL provides a standard-based interface to the platform that is easily accessible to internal and external user groups and easily extendable with plug-in components.

### 2.5.4 Synopsis

Based on the existing projects and tools, a flexible, extendable, multi-functioned configuration tool should:

- Enable verification for several router vendors

  The verification tool for multi-vendor network should verify across the router configuration files from various vendors.

- Enable users to add/create new configuration checking rules

  The rule set should be extensible for users to create new rules to validate configuration when new features are added to the devices.

- Combine new technologies into multi-vendor network management

  New techniques for data management have emerged. For example, XML, a new data description language that has changed Web design and data exchange, can be used as a general data repository for multi-vendor network management. The BlueBird project is a good example of how XML and XSL can be used in a network management system.

# Chapter 3

# Design

The main goal of our XML-based Verification Tool (XVT) is to implement a configuration verification tool for routers from different vendors. Compared with existing configuration verification tools mentioned in the chapter 2, XVT has the following functions in common:

- Parses the input configuration files.

  XVT provides the parser to process the input configuration files. The parser extracts the configuration information from the configuration files and uses the information to construct a global view, which is used for verification.

- Checks the configuration consistency errors following a predefined rule set.

  XVT checks and reports the configuration consistency errors within one specific router as well as across the routers in the network following predefined rules. The rules will be introduced in Section 3.1.2.

In addition to these functions in common with other tools, XVT contributes to multi-vendor network management as follows:

- Enables configuration verification across the multi-vendor network.

  There are many verification tools only designed for single-vendor networks, i.e., networks consist of routers from one specific company. OPNet's NetDoctor

is the most successful and well-known product implemented for multi-vendor network. But it is not extensible because users cannot easily add new features or update existing features of the system. In XVT new verification rules are easily defined using a small set of predefined operators.

- Uses XML to integrate network configuration data.

  A problem for network management of multi-vendor network is that the network components from different vendor companies use different standards. It would be easier to have an integrated view of network management capability by defining standards and using the standards across the multi-vendor network. However, network companies are unlikely to abandon their own existing standards and redesign their products following one universal standard. They prefer to use private standards to highlight the aspects of their products that differentiate them from their competitors. Furthermore, abandoning current standards is costly to redesign and update existing equipment.

  To integrate different standards requires the definition of a general data repository, or a universal network-wide view of these different configuration languages.

  It is desirable that the general data repository have the following characteristics [29]:

  - provide a single logical view.
  - provide a general naming structure so that the managed network objects can be easily stored and retrieved in a uniform way across a heterogeneous system.
  - provide a simple and efficient interface through which application programs or processes can easily access the management information stored in the repository.

- have international recognition so it has the potential to be accepted worldwide.
- provide reasonable performance so that the repository will not become the bottleneck.

The open issues for the general data repository include how to define it, how to represent it, and how to map the network information into it. We choose XML for the representation of the data repository. There will be more details about the reasons and the advantages of using XML to implement the data repository in section 3.2.1.

## 3.1 Parse and Check

### 3.1.1 Analysis of Configuration Files

There are various kinds of router products from different network device suppliers. Cisco is the most popular router company, its router products have 85.5% share of the global router market in the first quarter of 2002[30]. Juniper network company is the number two player behind Cisco which has 6.4% share of the global router market in the same period[30]. Because of their dominant roles in the market, Cisco and Juniper router configuration files were chosen for analysis since they represent a significant portion of the router products in the market. The goal of the analysis is to determine whether these two different configuration files can be mapped into one general data repository. The analysis was carried out for the comparison between the structure of the configuration files as well as the relationship among the configuration data. The prerequisite for the analysis is that the configuration files are to implement the same configuration tasks.

### Structure

Cisco and Juniper use a different configuration command format and keyword definition. But there still exist similarities between the two configuration formats that allow globalization.

The first similarity is the hierarchical structure of Juniper and Cisco configuration files. There is a hierarchical relationship among the configuration commands in both kinds of configuration files. Each router command has its own set of subcommands. Cisco router configuration files can be represented in a hierarchical tree with commands as the nodes and subcommands as the children nodes. Because Juniper's configuration files are represented in XML format, they also have a clear hierarchical structure. Hence Juniper configuration files can also be mapped into a hierarchical tree with several children nodes and leaves.

The second similarity is that the configuration commands of Cisco and Juniper configuration files can be grouped into the same sections. Cisco and Juniper router configuration files both consist of four major configuration sections: general settings, interface specification, filter specification, and routing protocol specification[8][9]. Each section contains commands performing basic configuration tasks. Following these sections, the commands in Juniper and Cisco router configuration files can be grouped into the same four paragraphs. Each paragraph contains related commands for one specific configuration section. In this way the configuration files of Cisco and Juniper can be mapped into one tree with the same children nodes and leaves.

### Configuration Data

The answer to whether different configuration data can be mapped into the same logical view also depends on whether there are certain logical relationships between the data. More specifically, with respect to the MPLS VPN configuration task, we compared Cisco and Juniper configuration files and found the following similarities:

- Configure the VPN instances on PE routers

  The VPN configuration examples in Figure 3.1 shows Cisco and Juniper both implement the same following VPN configuration commands on PE routers.

  - Define VPN routing instance

    Cisco uses the command "ip vrf vrf-name" to define the VPN routing instances while Juniper uses "<routing-instances> <instance> <name>" to implement the same task.

  - Create unique route distinguisher for each routing instance that has been configured.

    Cisco uses the command "rd rd-name" to define the route distinguisher while Juniper uses " <route-distinguisher><rd-type>rd-name" to implement the same task.

  - Configure the import or export policy

    Cisco uses the command "route-target export export-policy-name" and "route-target import import-policy-name" to define the import or export policy while Juniper uses "<vrf-import><name> import-policy-name" and "<vrf-export><name> export-policy-name" to implement the same task.

  In addtion to the VPN configuration on PE routers, there exist similaries for the VPN configuration tasks on PE, P and CE routers.

  - Configure BGP routing sessions between PE routers

    As showed in Figure 3.2, Cisco uses "address-family vpnv4" and "neighbor" commands to set up the BGP session between the PE routers and their BGP neighbors while Juniper uses "<bgp><neighbor>" to implement the same task.

  - Configure VPN routes between PE and CE routers

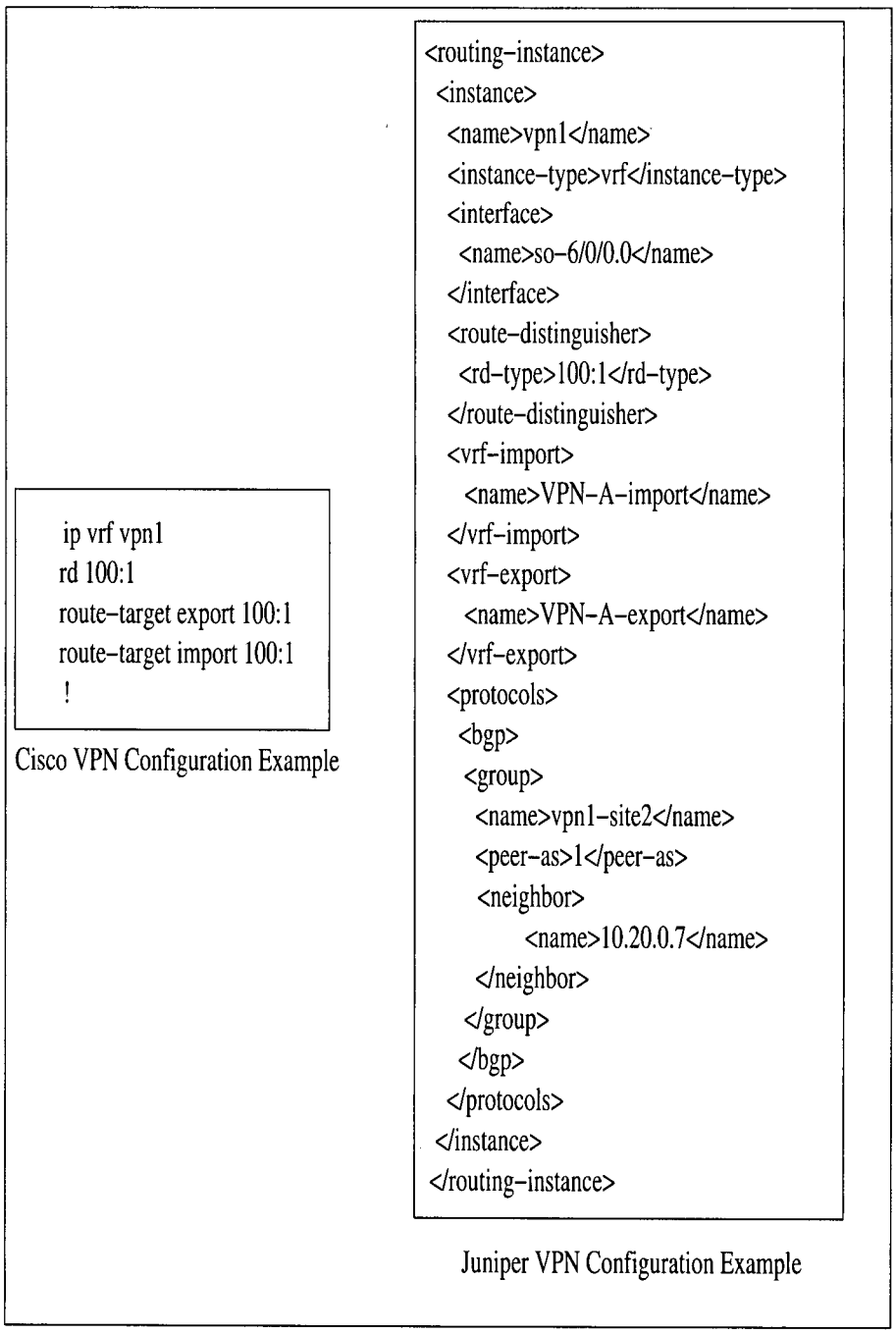



Figure 3.1: Cisco and Juniper's MPLS VPN Configuration File Sample

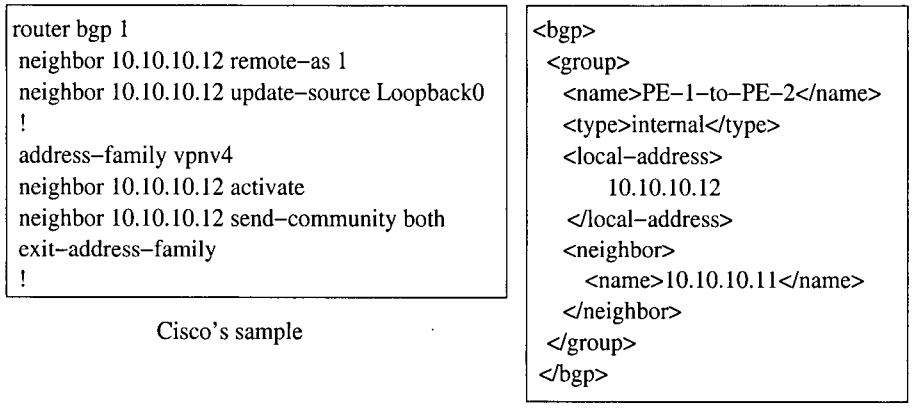



Figure 3.2: Cisco and Juniper's VPN BGP Session Configuration File Samples

Cisco uses "address-family ipv4 vrf vpn1" and "neighbor" commands to set up the BGP session between PE and CE routers and the BGP neighbors while Juniper uses "<protocol><bgp><group><neighbor>" to implement the same task.

- Configure MPLS
  - Enable MPLS protocol on the routers
  - Configure the explicit paths and Label Switched Path(LSP)

As shown in Figure 3.3, Cisco and Juniper both list the hops in the route to configure the named path. Cisco uses "next-address" while Juniper uses "<path-list><name>" to add hops into the named path.

After the configuration of explicit paths, Cisco and Juniper configure the LSPs following the same principle: choose one path from the defined explicit paths as a LSP. As shown in Figure 3.4, Cisco uses "tunnel mpls traffic-eng path-option 1 explicit identifier explicit-path-name" to configure the LSP while Juniper uses "<label-switched-path><name>" to implement the same task.

```
ip explicit-path identifier 1
 next-address 131.0.1.1
 next-address 131.0.2.1
 next-address 131.0.3.1
 next-address 131.0.6.5
 next-address 131.0.7.1
 next-address 131.0.13.1
 next-address 131.0.14.2
 next-address 131.0.16.1
!
```

```
<path>
  <name>pe2-p9-p4-pe1</name>
  <path-list>
   <name>131.0.16.2</name>
   <strict/>
  </path-list>
  <path-list>
   <name>131.0.12.3</name>
   <strict/>
  </path-list>
  <path-list>
   <name>131.0.10.3</name>
   <strict/>
  </path-list>
  <path-list>
   <name>131.0.4.2</name>
   <strict/>
  </path-list>
  <path-list>
   <name>131.0.2.2</name>
   <strict/>
  </path-list>
  <path-list>
   <name>131.0.1.2</name>
   <strict/>
  </path-list>
</path>
```

Figure 3.3: Cisco and Juniper's Explicit Path Configuration File Samples

```
interface tunnel 1
  ip unnumbered S1/2
  tunnel destination 10.10.10.12
  tunnel mode mpls traffic-eng
  tunnel mpls traffic-eng badnwidth 100
  tunnel mpls traffic-eng priority 1 1
  tunnel mpls traffic-eng path-option 1 explicit identifier 1
!
```

Cisco Label Switched Path Configuration Example

```
<label-switched-path>
  <name>lsp1</name>
  <to>10.10.10.11</to>
  <hop-limit>32</hop-limit>
  <bandwidth>10m</bandwidth>
  <primary>
   <name>pe2-p9-p5-p1-pe1</name>
  </primary>
 </label-switched-path>
```

Juniper Label Switched Path Configuration Example

Figure 3.4: Cisco and Juniper's LSP Configuration File Samples

- Cisco and Juniper both implement the same tasks using similar commands for other basic configurations for (1) interface configuration (2) global settings configuration and (3) protocol configuration.

### Conclusion of Analysis

Based on the same MPLS VPN architecture and a subset of the configuration commands for the same configuration tasks from Cisco and Juniper configuration files, the following conclusions can be drawn from the analysis:

- It is possible to have one general data repository for Cisco and Juniper configuration languages

  Each configuration file can be seen as a local view of each router in the network. Although these local views are represented in different languages, each can be represented in the same tree structure with similar configuration data as the content.

- Different configuration languages can be mapped into the general data repository

A conceptually unified network-wide view of the data is permitted although the data are heterogeneous. The configuration data in the local configuration files will be mapped into the global data repository based on its configuration category and functionality. For example, the data for interface IP address configuration is mapped into the IP address field under interface category in the global view.

### 3.1.2 Error Checking Rules

There are two kinds of variables in the configuration files: those that do not need any knowledge of other variables and those that do. The dependencies between the variables can be classified into the dependency within one router configuration file as well as the dependency across the router configuration files.

For the dependency within one router configuration file, the variables reference variables in the same router configuration file. The dependencies within a file can help to identify the relationship between interfaces, protocols, and other static route objects. For example, an interface may be enabled with specific protocols, and configured with a set of static routes.

There are two kinds of dependencies within one router configuration file—the dependency within one section of the configuration file and the dependency across the sections of the same configuration file. The section refers to the categories that the configuration commands are grouped into, for example, global settings.

The dependency across the router configuration files can be exploited to derive the links that represent the physical and logical connectivity between the routers. The dependency across the routers also includes a consistent definition of variables and consistent reference to remote nodes. Any inconsistency of these two dependencies breaks the relationship between the routers and may lead to network problems. For example, if the link between an ingress router and its next hop in the LSP fails to be constructed, the traffic cannot go into MPLS network because

of the broken link.

Based on the analysis of the relationship among the variables in the configuration files, the error checking rules can be classified into four categories as independent variables, dependent variables within the section of the same router configuration file, dependent variables across different sections within the same router configuration file and dependent variables across different router configuration files.

## 3.2 Multi-vendor Network Management

### 3.2.1 Global View

#### Data Model

The general data repository, or the global view, is so important that parsing, checking, and querying processes are all based on it. The requirements of the data model of the global view should be:

- Flexible structure

  Since we are considering large, multi-vendor networks with devices using different configuration languages, the global view that represents the information of this network should not be limited to the format following one vendor's standard. Moreover, it should have a flexible structure to ease the mapping of the various types of device files into the global view.

- Easily manipulated

  The data structure of the global view should make the global view easy to access and manipulate, otherwise it may be computationally expensive to parse, check and query the global view.

- Easy for transformation

Transformation is important because the format of the error report may vary depending on the user's requirement. For example, some network operators may require an HTML file as the output error report so that users can browse the error report online, while other operators may need an error report in plain text so that it can be saved as the error log file. As a result, easily transforming from one format into another one is important.

- Convenient for query

  Users can obtain useful information by querying the configuration information that is not showed in the error report. Hence the global view should be easily accessed and queried.

  One approach is to define the global view using relational database tables. Relational database tables meet the "convenient for query" requirement but do not meet other requirements because the structure of the relational database files is fixed. It does not allow arbitrary elements. However for network management data there should be flexibility in the number of data elements, for example, the number of interfaces of one specific router. Another disadvantage of relational database tables is that the difficulty in changing the database schema.

Lastly XML is chosen as the representation of global view because:

- XML is a language for "semi-structured data" in the sense that many different formats of data can be fixed in the same schema. Furthermore XML places no restriction on tags, attribute names or nesting patterns. Hence XML meets the requirement of being a "flexible structure".

- The format of XML document is hierarchical rather than relational or object-oriented. Each XML file can be represented as a tree with one root and several levels of nodes that are the elements in the XML file. The tree data structure

is simple and easy to manipulate as a basic data structure. There also are many commercial parser tools for XML documents. In this sense XML is easily manipulated by software. So XML meets the requirement of "easily manipulated".

- There is a XML related technology named XSLT for XML transformation. There are a few successful XSLT engines, for example, Xalan[23], distributed to programmers. The only task needed for transformation is writing the transformation rules in XSLT. The source XML documents can be transformed into HTML, XML, or plain text files by applying XSLT to the XML document. So XML meets the requirement "easy for transformation".

- There are several XML-based query languages, for example, XQuery[24] and XQL[25]. For some of these query languages there are a few query engines available. The query languages are based on XML Path Language (XPath) that uses simple path information to address specific elements in XML documents. So XML meets the requirement "convenient for query".

  Although XML is a tree-structured language that is not suitable to represent the graph-based network, and XML query technology is not as mature as database queries, XML's flexibility and the various tools that are available make XML a reasonable choise.

### Content

The second step is to decide how the configuration data is put into the global view to correctly represent the collection of configuration data of the whole network. A naive approach is to put all local variables and their values into the global view to construct a superset of the configuration files. But there exist some variables from different configuration files that define and describe the same configuration data while using different keywords. Only collecting information together into one

file without any further processing cannot help to find the relationship between the configuration data. The data is only syntactically collected but not semantically collected.

The analysis of the configuration files has shown that for the same configuration task there exists a lot of similar configuration commands although they use different keywords. For example, to enable MPLS on the router, Cisco uses the "ip cef distributed" command while Juniper uses the "family mpls" command. Hence the collection of the keywords from the configuration files can be condensed to a universal keyword set for the global view. The semantic-related keywords that implement the same configuration task are mapped into the same keyword in the global view. For example, the variable "LSP" in Juniper and the variable "tunnel" in Cisco are mapped into the keyword "LSP" in the global view. "ip cef distributed" from Cisco and "family mpls" from Juniper are mapped into the keyword "mplsenabled".

The keywords from the universal keyword set construct the XML tags for the global view. The values for the tags are the configuration data values from each individual configuration file.

The tag definition of the global view in the XML document is represented in Figure 3.5. There are six sections in the global view, each section is represented by matched 2-level tags. The "global" element is the 1-level tag. The configuration data about the global setting is collected under the "hosts" tag. The name and IP address of the router, the vendor's name, the setup of the path set in the network, etc. are under the "hosts" tag. The data under the "interfaces" tag configures the interface. The IP address and prefix of the interface as well as the enabled protocols on individual interfaces appear under the "interface" tag. Under the "lsps" tag is the data about label switched path information including the destination of the path and its path name. The path name is referenced from the path set in the global setting under the "hosts" tag. The "VPNS" tag is for MPLS VPN configuration. The data under the "VPNS" tag includes the data for the import and

```
<?xml version="1.0" encoding="UTF-8"?>
<global>
 <hosts>
  <host>
   <hostname></hostname>
   <ip />
   <type></type>
   <mplsenabled></mplsenabled>
   <path name="">
    <address />
   </path>
  </host>
 </hosts>
 <interfaces>
  <interface name="" hostname="">
    <ipaddress></ipaddress>
    <submask></submask>
    <protocol></protocol>
  </interface>
 </interfaces>
 <lsps>
  <lsp name="" hostname="">
    <destination></destination>
    <pathname></pathname>
  </lsp>
 </lsps>
 <VPNS>
  <VPN name="" hostname="">
   <interface></interface>
   <distinguisher></distinguisher>
   <import></import>
   <export></export>
   <bgp>
    <vpnname/>
    <neighbor />
   </bgp>
  </VPN>
 </VPNS>
 <bgps>
  <bgp>
    <hostname />
    <neighbor />
  </bgp>
 </bgps>
 <topo>
  <router host="">
    <neighbor/>
  </router>
 </topo>
</global>
```

Figure 3.5: Global View

export policy, distinguisher, etc.. The "bgps" tag is for BGP protocol configuration. The data under this tag mainly consists of the BGP neighbor information. The last tag is "topo". It is the section describing the links between routers and the topology of the network. All data except topology information directly comes from the original configuration file. The topology information is constructed based on the configuration data of the interfaces. If the configuration data of the interfaces changes, the topology information may change as well.

The global view shown in Figure 3.5 has the following characteristics:

- provides a single logical view.
- provides a general naming structure so that the managed configuration data can be easily stored and retrieved in a uniform way across the heterogeneous system.
- provides a simple and efficient interface through which application programs or processes can easily access the management information stored in the repository.
- provides reasonable performance so that accessing the repository is not the bottleneck.

### 3.2.2 Keyword Set

The keyword set is designed to define the mapping rules when the local keywords are mapped into the keywords in the global view. How the keywords are defined and how the mapping rules are defined is discussed in this section.

#### Keyword

The structure of the keyword set is a hierarchical tree. The configuration categories, such as global settings, interfaces, and protocols, are represented by different

```
<!ELEMENT keywordset(keywords)>
<!ELEMENT keywords(type,subset+)>
<!ELEMENT type(#PCDATA)>
<!ELEMENT subset(global, interface, lsp, vpn, bgp)>
<!ELEMENT global(keyword*)>
<!ELEMENT interface(keyword*)>
<!ELEMENT lsp(keyword*)>
<!ELEMENT vpn(keyword*)>
<!ELEMENT bgp(keyword*)>
<!ELEMENT keyword(#PCDATA)>
```

Figure 3.6: DTD of Keyword Set

nodes in the tree. For the original configuration files with the similar hierarchical tree structure, the mapping of the structure between the global view and local configuration files is easy because the mapping is between two similar trees. For the original configuration files with no clear tree structure, they need to be mapped into the tree-structured global view. The mapping procedure is to collect configuration data in the original configuration files under specific configuration categories in the global view.

The keyword set is presented in XML format. A Document Type Definition (DTD) for the keyword set is showed in Figure 3.6, and is shown pictorially in Figure 3.7.

In XVT there are two coded sets of keywords, one for Cisco and the other one for Juniper. There are two types of commands in Cisco configuration files. The first type of commands assigns values to the variables and follows "keyword value" format. For example, "hostname p1" is the command to configure the router name. The keyword is "hostname" and the value is "p1". The second type of commands enables conditional variables and follows "keyword" format. The keyword for this kind of commands is the whole command and the data value is a boolean value. For example, "tag-switching ip" is the command to enable MPLS protocol on one

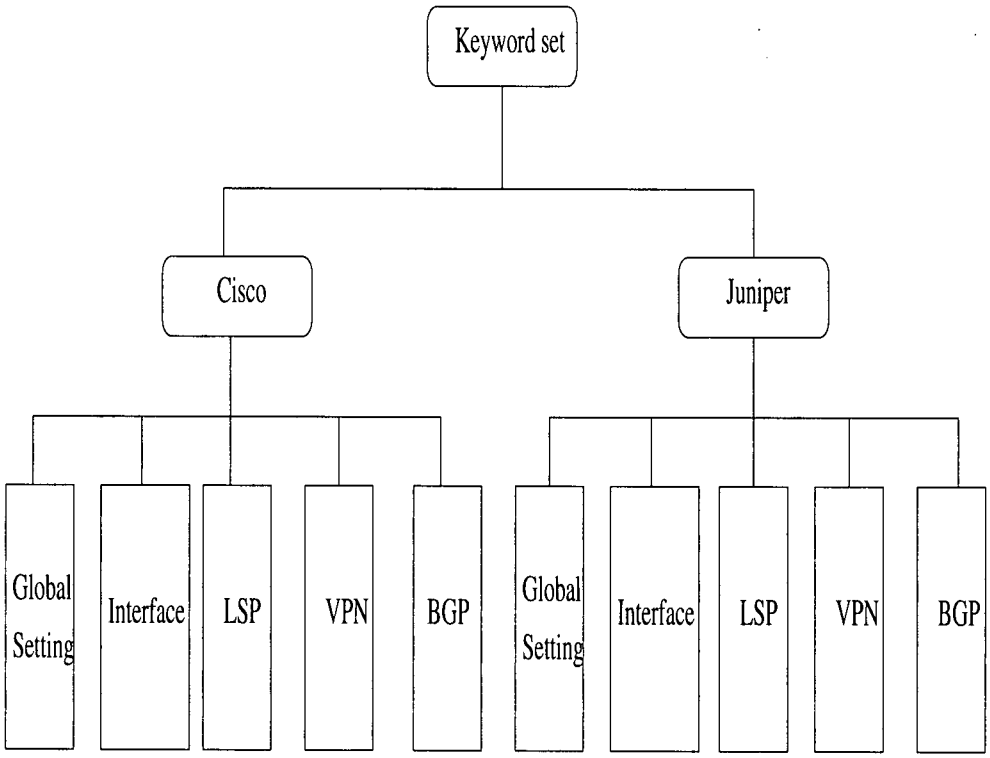



Figure 3.7: The Structure of the Keyword Set

specific interface. The keyword is "tag-switching ip" and the data value is "true".

The keywords in the Juniper configuration files cannot be the single tag name from one element because XML file is context-sensitive. Hence the keyword definition includes the path information to indicate the tag's ancestors that represent the configuration categories. For example, Table 3.1 is a Juniper configuration file to configure the protocols. The same "interface" keyword in Table 3.1 under different tags means different interface for different protocols. The keywords "protocols/mpls/interface" and "protocols/ldp/interface" define the interfaces for two different protocols. "protocols/mpls/interface" means the "interface" is the child element of "mpls" and the descendant of "protocols".

```
<protocol>
  <mpls>
    <interface>
      <name>s0-0/0/0/0</name>
    </interface>
  </mpls>
  <ldp>
    <interface>
      <name>s0-0/0/1/0</name>
    </interface>
  </ldp>
</protocol>
```

Table 3.1: Juniper's Configuration File Example

### Mapping Rules

The original configuration files are parsed sequentially using the keywords in the keyword set. The parsing procedure gets the data value for each keyword and constructs the pair <local keyword, data value>. The local keyword is mapped to the keyword in the global view based on its configuration category. For example, the local keyword "neighbor" under the "VPN" category, which is used to configure the neighbor in VPN, is mapped to the keyword representing VPN neighbor in the

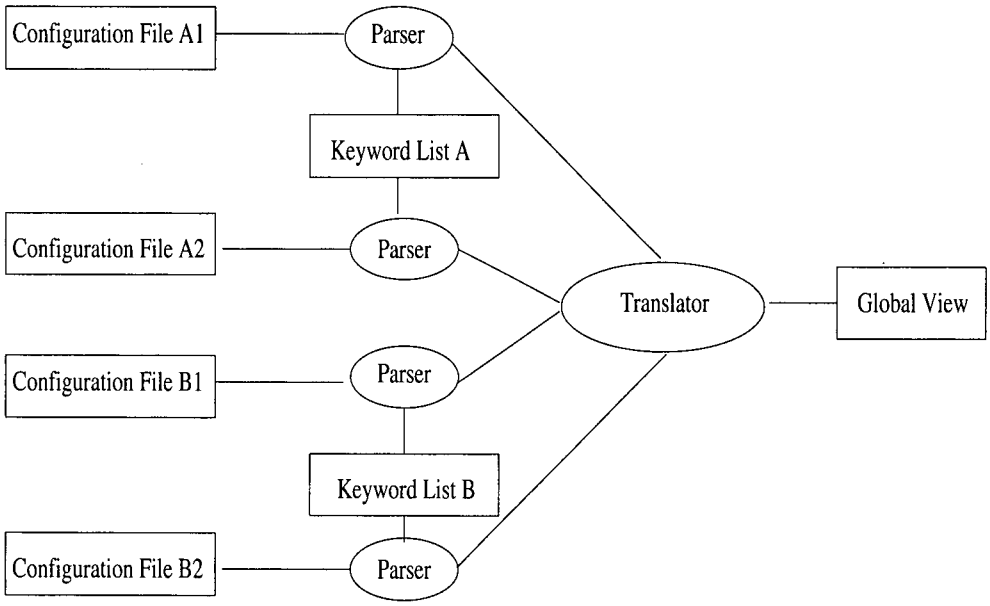



Figure 3.8: Parsing and Mapping Procedure

global view, while the local keyword "neighbor" under "BGP" category which is used to configure the neighbor in BGP, is mapped to the keyword representing BGP neighbor in the global view. The data value is assigned to the element with the global keyword as the tag in the global view. The updated global view is saved. Figure 3.8 is an illustration of the procedure.

### Advantages

The advantages of using keyword set are its simplicity, its extensibility, and its ease for evolution. Network technology evolves quickly. The introduction of new concepts always triggers new features to be added into the configuration files. The keyword set allows new keywords to be easily added to incorporate new features as they are introduced.

### 3.2.3 Query

How to define the user query format is discussed in this section.

One extreme for queries is to use the XML-based query engine's syntax. Using the query engine's syntax reduces the processing time because users' queries are not transformed into machine accessible format. Using the query engine's syntax also reduces programming labor because there are query engines available. But users are required to be familiar with the query engine's syntax.

Another extreme is to use natural language for queries. The query processing is too complicated in this case.

A middle approach is for users to follow the syntax of a higher level meta language. The requirements of this meta language are:

- easy to use

  The meta language should be small and easy for users to learn..

- network-oriented

  Since the application is for network management the meta language should be equipped with network-oriented features, for example, graph-based queries.

We chose to implement an operator-oriented meta language as the query language. Users do not need to worry about the lower level query process. They can write the query requests using the provided operators in the same way they write mathematical formula. In this way, users can easily understand the query and it is not difficult to become familiar with the few operators in the language.

# Chapter 4

# Implementation

In this chapter the details of the implementation of XVT tool will be introduced. XVT tool is decomposed into several modules as shown in Figure 4.1. These modules will be explained in detail.

- Parser

  The parser takes the configuration files and the pre-defined keyword set as input. The parser then parses the configuration files to obtain the value for each keyword in the predefined keyword set.

- Translator

  The translator maps the keyword from the local configuration file into the keyword in the global view. In addition, it assigns the global keyword the value of the local keyword.

- Topology Extractor

  The extractor extracts the topology information including connectivity information from the global view. A graphical representation of the topology is shown.

- Checker

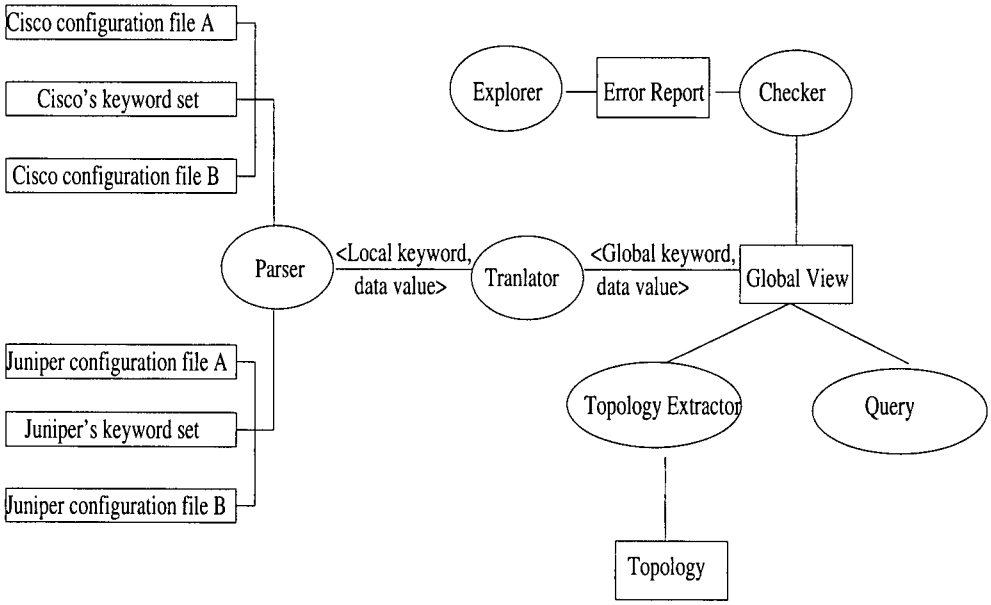



Figure 4.1: Modules of XVT

The checker checks the configuration errors within one router as well as across routers. The errors are saved in an error report.

- Explorer

  The explorer allows users to browse the final error report. Users can browse the error report by using a browser if the error report is an HTML file.

- Query

  The query enables users to access to the configuration information that is not provided by the error report. For example, users can query the name of routers that are configured with MPLS protocol.

```
<keywords>
  <cisco>
    <global>
        <keyword>hostname</keyword>
        <keyword>ip address</keyword>
        <keyword>ip cef</keyword>
        <keyword>explicit-path identifier</keyword>
    </global>
    <interface>
        <keyword>interface</keyword>
        <keyword>ip address</keyword>
    </interface>
  .....
  </cisco>
</keywords>
```

Figure 4.2: Example of Keyword Set

## 4.1 Parser

The parser reads through the configuration files to parse the file following the order of keywords in the predefined keyword set. The keywords are predefined following the router's configuration documentation. Each router vendor may have different keyword sets because it uses a different configuration language.

The keywords in the keyword sets are parsed sequentially. Once the keyword is found in the configuration file, the value for the matched keyword is fetched from the original configuration file. The <keyword, value> pair is returned to "Translator" module for further parsing. This process repeats until all the keywords in the keyword sets are extracted from the original configuration file.

An example of Cisco keyword set is shown in Figure 4.2. In this example there are four keywords in the configuration file that are parsed for configuration data as part of the global settings. The parser first searches the keyword "hostname" in the configuration file. Once it is found, the parser looks up the value for keyword "hostname" from Table 4.2. The parser continues to search for the next keyword

"ip address" after keyword "hostname" is parsed. The process continues until all keywords in the keyword set are parsed.

For Juniper configuration files, the single keyword cannot indicate one specific configuration command as was discussed in Section 3.2.2. Hence the keyword for Juniper needs to include the path information. For example, the keyword for the IP address configuration is "system/inet" to indicate that the IP address is configured under "inet" tag with "system" as the parent tag. The parser uses XSLT to parse the keyword, extract the value for the "system/inet" keyword and output the pair <system/inet, data value>.

| Configuration File | Keyword | Output pair |
|---|---|---|
| <system><br><host-name> x<br></system> | system/host-name | <system/host-name, x> |
| <system><br><inet> x<br></system> | system/inet | <system/inet, x> |
| <protocols><br><mpls><br></protocol> | protocols/mpls | <protocols/mpls, yes> |
| <system><br><path><br><name/>x<br><path-list><br></name><br>y1<br></name><br></path-list><br></path><br></system> | system/path/name<br><br><br><br><br><br>system/path/path-list/name | <system/path/name, x ><br><br><br><br><br><br><system/path/path-list<br>/name, y1> |

Table 4.1: Parsing Global Setting Variables for Juniper Configuration Files

| Configuration File | Keyword | Output |
|---|---|---|
| hostname x; | hostname | <hostname,x> |
| ip address x; | ip address | <ip address, x> |
| ip cef; | ip cef | <ip cef, yes> |
| ip explicit-path identifier x; | ip explicit-path identifier | <ip explicit-path identifier,x> |

Table 4.2: Parsing Global Setting Variables for Cisco Configuration Files

## 4.2 Translator

The translator obtains the pair <local keyword, data value> from the parser and maps the local keyword into the keyword in the global view.

The input to the translator module is the pair <local keyword, data value>. The translator maps the local keyword into the keyword in the global view based on the mapping rules that will be explained next. The translator saves the value under the global keyword field found in the global view.

The mapping rules are given in Table 4.3 to Table 4.7. Different tables represent different categories of keywords.

| Global View | Juniper | Cisco |
|---|---|---|
| <host><br><hostname/>x | <system/host-name,x> | <hostname,x> |
| <ip/>x | <system/inet,x> | <ip address,x> |
| <mplsenable/> yes | <protocols/mpls,yes> | <ip cef,yes> |
| <path name=x ><br><br><address><br>y1<br></address><br></path><br></host> | <system/path/name,x><br><br><system/path/path-list /name,y1> | <ip explicit-path identifier,x><br><next-address,y1> |

Table 4.3: Mapping for Global Setting Variables

There are two different addressing schemes. The traditional scheme is that

| Global View | Juniper | Cisco |
|---|---|---|
| <interface name=x> | <interfaces/interface/name, x> | <interface,x> |
| <protocol/>mpls | <interfaces/interface/unit/family mpls,yes> | <tag-switching ip, yes> |
| <ipaddress><br>x<br></ipaddress><br><prefix>y</prefix><br></interface> | <interfaces/interface/unit/ family/inet/address/name,x/y> | <ip address,x/y> |

Table 4.4: Mapping for Interface Variables

the addresses are assigned into three classes: class A, class B and class C. The prefix for classes A, B and C are 8, 16 or 24 bits. Another scheme is called Classless Inter-Domain Routing (CIDR)[31] that is a replacement for the old process of assigning class A, B and C addresses with a generalized network prefix. CIDR currently uses prefixes from 13 to 27 bits. The prefixes are represented by using integer numbers. For example, in the CIDR address 206.13.01.48/25, "25" indicates the first 25 bits are used to identify the unique network.

For these two addressing schemes the translator translates the prefix of the interface into the common 32-bit format. For example, for the interface configured as 144.103.1.3/25, the value 25 is mapped into 255.255.255.1.

| Global View | Juniper | Cisco |
|---|---|---|
| <lsp name=x> | <label-switched-path/name,x> | <interface tunnel,x> |
| <pathname/>x | <label-switched-path/primary/ name,x> | <tunnel mpls traffic-eng path explicit identifier,x> |
| <destination/>y<br></lsp> | <label-switched-path/to,y> | <tunnel destination,y> |

Table 4.5: Mapping for MPLS Label Switched Path Variables

An example is given to demonstrate how the translator works. The input is a text line from a Cisco configuration file, say, "route-target export 100:1". The translator extracts the pair <route-target export, 100:1> from the parser. Following

| Global View | Juniper | Cisco |
|---|---|---|
| <VPN><br><vrfname/>x | <routing-instances/instance/<br>name,x> | <ip vrf,x> |
| <distinguisher/>x | <routing-instances/instance/<br>route-distinguisher,x> | <rd,x> |
| <import/>x | <routing-instances/instance<br>vrf-import/name,x> | <route-target<br>import,x> |
| <export/>x | <routing-instances/instance/<br>vrf-export/name,x> | <route-target<br>export,x> |
| <bgp><br><vpnname/>x<br><neighbor/>y<br><br><br></bgp><br></VPN> | <routing-instances/instance/<br>protocols/bgp/group/name,x><br><routing-instances/instance/<br>protocols/bgp/group/neighbor/<br>name,y> | <address-family<br>ipv4 vrf,x><br><neighbor<br>remote-as<br>as-system-number,<br>y> |

Table 4.6: Mapping for MPLS VPN Variables

| Global View | Juniper | Cisco |
|---|---|---|
| <bgp><br><neighbor/>x<br></bgp> | <protocol/bgp,x> | <neighbor activate,x> |

Table 4.7: Mapping for BGP Variables

the mapping rules given in Table 4.6 the translator finds that the matched keyword is <export> in the global view represented by XML. As a result the data value under the "export" tag in the global view is changed to 100:1. Finally the translator saves all the changes to the global view.

## 4.3 Checker

The checker follows predefined rules for error checking. The checking rules are classified into three categories: within the section of the same router configuration file, across different sections within the same router configuration file, and across

different router configuration files.

### 4.3.1 Within the Section of the Same Router Configuration File

Checking the variables within a section is separated into two categories, independent checking and dependent checking. Independent checking occurs whenever the process does not have to be compared with any other variables. In the case of dependent variables it is necessary to compare their values with other values in the configuration files.

#### Independent Checking within a Section

Independent checking within a section includes:

- Whether the variable value is a valid IP address

  An IP address is a 32 bit binary number represented as four decimal values, each representing eight bits, separated by decimal points, i.e., x.x.x.x. Each decimal value should be in the range 0 to 255 (known as octet).

  The specific variables checked for this rule include:

  - IP address for each interface
  - IP address for each router
  - IP address for each hop in the explicit path
  - IP address for the neighbor configuration in BGP protocol
  - IP address for the "destination" address for a label switched path
  - IP address for the neighbor configuration in an VPN BGP section

  Although Juniper and Cisco routers can check the format of IP address, checking IP address is valid or not is implemented for the worst case that routers do not provide such checking.

- Whether the IP prefix is defined valid

  The IP prefix is represented in the same way as IP. So the format of the IP prefix should be as x.x.x.x. Each decimal value in the IP prefix cannot be greater than 255.

  The specific variables for this rule include:

  - Prefix for each interface

- Whether the keywords are assigned a value

  XVT generates the error if the value field in the pair <keyword, value> is null. This error occurs when a keyword in the configuration file has not been assigned a value and may indicate a problem. For example, the "destination" field cannot be null for a label switched path if MPLS protocol is enabled on the router.

### Dependent Checking within One Section

Dependent checking within one section includes:

- Checks the uniqueness of variables

  Some variables are required to be assigned a unique value. For example, if two interfaces are assigned the same name "s0-0/0/0.0" on the same router but with different protocols, the router does not know which protocol to follow if packets are forwarded through the interface. These kind of errors are checked within each section of the configuration files. For example, checking within the "interface" section to see whether there are two interfaces are assigned the same name.

  The specific variables for this rule include:

  - Name of the interfaces within each router

- Name of the Label switched paths within each ingress router
- Name of VPNs within each router
- Name of the explicit paths within each router

### 4.3.2 Across Different Sections within the Same Router Configuration File

The consistency check across different sections within the same router configuration file includes:

- Whether the referenced variables are consistent

  The meaning of "referenced" is that the value of "referenced" variables should come from a predefined value set. If the referenced value is not equal to any in the value set, there are consistency problems. For example, both Juniper and Cisco follow the same rules to define an explicit label switched path: they configure a set of named paths with its name and hops first in the global setting section, then they configure the explicit path by referencing the path name from the defined named path sets. When the router requires the next hop address before the label is assigned, it first gets the explicit path name from the configuration file. If the explicit path is defined, then it searches the path set in the global setting section to get the matched named path. Following the configuration of the named path the router gets the next hop information. If the explicit path name is not referenced from the path set, the packets cannot be forwarded to the next hop.

  The specific variables for the reference rule include:

  - LSP name in the MPLS "protocol" section should be referenced from the path set in the "global setting" section

- Whether the protocols and related attributes are configured correctly.

The rules for protocols and attributes include:

- If MPLS protocol is not enabled on the router in the "global setting" section, the MPLS-related attributes in the "protocol" section cannot be assigned any value.
- If MPLS protocol is enabled in the "global setting" section, there should be at least one label switched path configured in MPLS "protocol" section. The attribute "destination" and "label switched path name" in MPLS "protocol" section also should be assigned the value.

### 4.3.3 Across Different Router Configuration Files

The consistency check across different router configuration files includes:

- The links between the hops in the label switched path should be set up by the interface configuration. For example, If A is configured as B's next hop in the named path, router A and router B should have the interface configuration to set up the corresponding link.
- BGP neighbor should be consistent. If A's BGP neighbor is B, then B's BGP neighbor should be configured as A.
- The hops indicated in the label switched path should match the router IP address or interface address.

### 4.3.4 Implementation of Rules

In XVT, all rules except IP/prefix address and link check are coded in XSLT. The reason why these two rules are coded separately is that XSLT does not have a "bit wise and" operator. The reason of using XSLT is for easy transformation to produce a final error report as well as its XML-based query capability.

```
<xsl:for-each select="VPNS/VPN">
  <xsl:variable name="hostname"><xsl:value-of select="@hostname"/></xsl:variable>
  <xsl:variable name="vpn"><xsl:value-of select="@name"/></xsl:variable>
  <xsl:if test="count(following::VPN[@hostname=$hostname and @name=$vpn])>=1">
    <error>
      <rule> Rule2:check invalid unique variables </rule>
      <type> dependent variable </type>
      <description>The VPN name <xsl:value-of select="$vpn"/> in the router
                  <xsl:value-of select="$hostname"/> should be unique
      </description>
    </error>
  </xsl:if>
</xsl:for-each>
```

Figure 4.3: Example of Rules Defined in XSLT

Figure 4.3 is the rule defined in XSLT to discover the variable "VPN name" that violates the uniqueness requirement. If there is more than one VPN name that is defined within one router, the error is reported and the output consists of the type of errors (dependent or independent variables), the rule it violated and the place where the error occurs. The error is saved in a temporary XML file and the XML file is later transformed into an HTML formatted error report.

## 4.4 Topology Extractor

The topology extractor obtains the network topology information from the global view. The extracted information includes connectivity for IP networks.

The extractor determines the connectivity for the IP network based on the interface configuration data. If two interfaces have the same subnet, there is a link between these two interfaces. After searching all links, the network topology is drawn to represent the node and link information. The output of the topology extractor is shown in Figure 4.4.

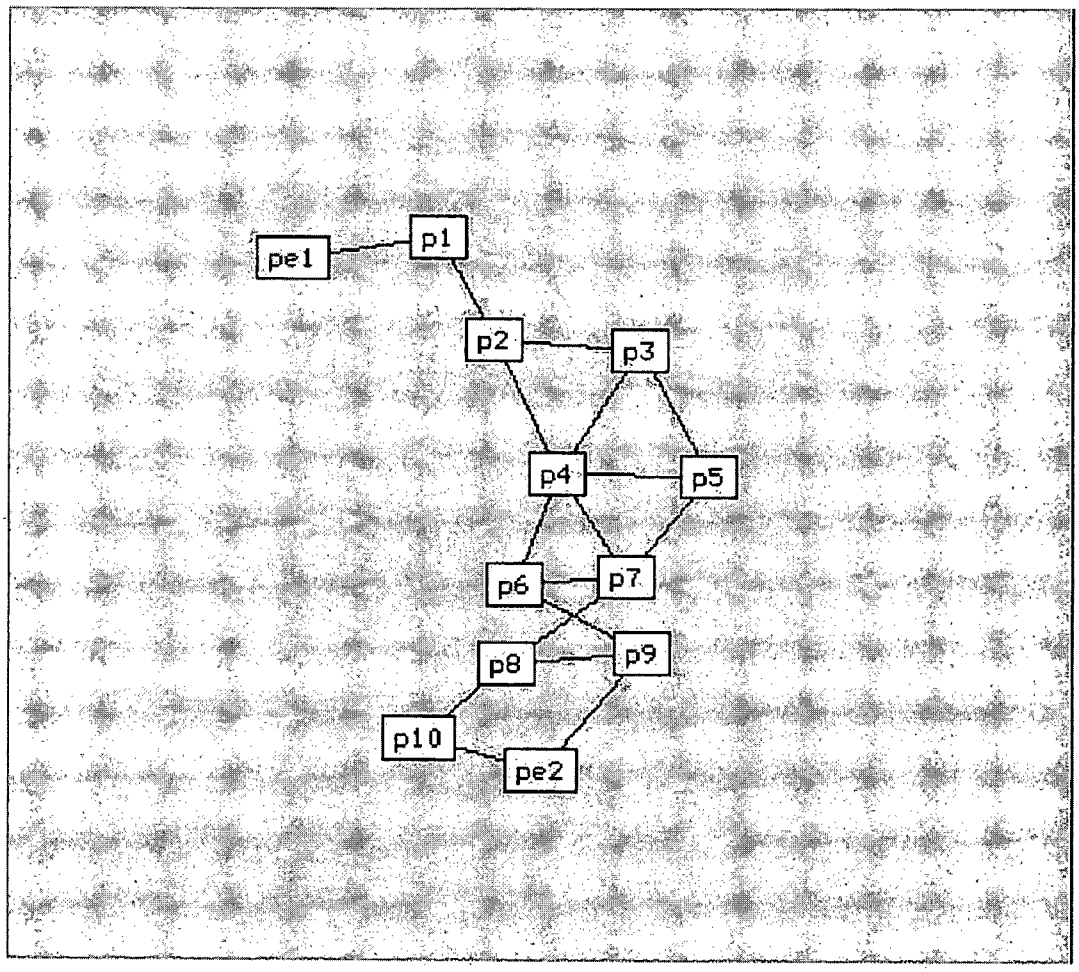



Figure 4.4: Topology

BGP connectivities are determined by BGP neighbors in the BGP protocol configuration. Connectivity for BGP protocol is not shown in the topology graph.

## 4.5 Explorer

The explorer is used to show the final error report. The error report is in HTML format and can be viewed in an HTML browser. Figure 4.5 is the example of the error report. It consists of the violated rules, error type and the place where the error occurs in the individual configuration file.

## 4.6 Query

Since the explorer only provides configuration error information about the network, query is used to give users access for more network information. Furthermore the query operators help users to create new verification rules.

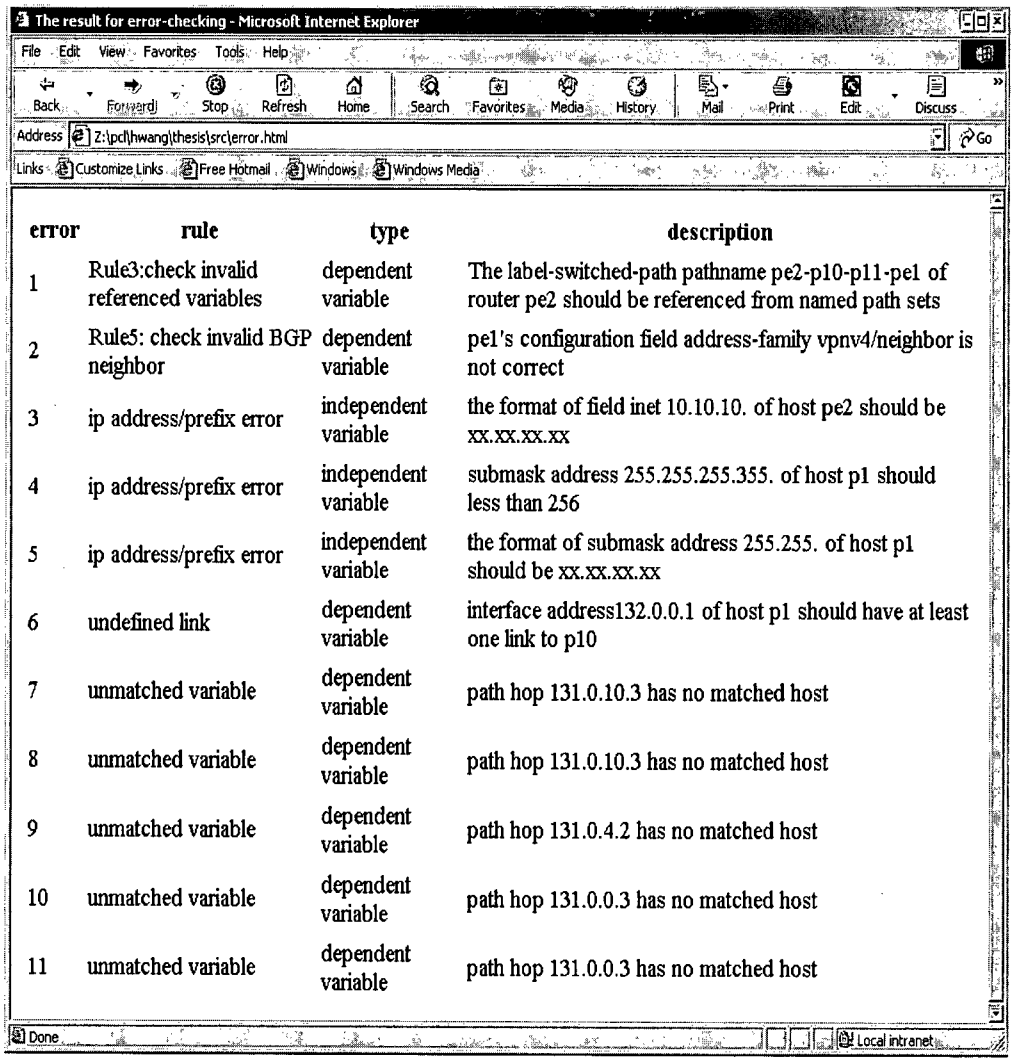
### 4.6.1 The Query Language

A meta query language is designed to help simplify user queries. The meta language is based on a few operators so that the user can query the global view by using a combination of given operators. The Command Line Interface (CLI) and a simple graphical interface both can be used for the query.

An Extended Backus-Naur Form (EBNF) for the query language is showed in Figure 4.6.

#### Overview

The predefined operators are listed in Table 4.8. Since XML is a tree-structured language that is not suitable to present the graphs, the operator "nodeAdj" is designed for graph-based querying. The "nodeAdj" command provides two graph-based views of the network: one is the BGP network graph and the other is the MPLS network




| error | rule | type | description |
|---|---|---|---|
| 1 | Rule3:check invalid referenced variables | dependent variable | The label-switched-path pathname pe2-p10-p11-pe1 of router pe2 should be referenced from named path sets |
| 2 | Rule5: check invalid BGP neighbor | dependent variable | pe1's configuration field address-family vpnv4/neighbor is not correct |
| 3 | ip address/prefix error | independent variable | the format of field inet 10.10.10. of host pe2 should be xx.xx.xx.xx |
| 4 | ip address/prefix error | independent variable | submask address 255.255.255.355. of host p1 should less than 256 |
| 5 | ip address/prefix error | independent variable | the format of submask address 255.255. of host p1 should be xx.xx.xx.xx |
| 6 | undefined link | dependent variable | interface address132.0.0.1 of host p1 should have at least one link to p10 |
| 7 | unmatched variable | dependent variable | path hop 131.0.10.3 has no matched host |
| 8 | unmatched variable | dependent variable | path hop 131.0.10.3 has no matched host |
| 9 | unmatched variable | dependent variable | path hop 131.0.4.2 has no matched host |
| 10 | unmatched variable | dependent variable | path hop 131.0.0.3 has no matched host |
| 11 | unmatched variable | dependent variable | path hop 131.0.0.3 has no matched host |

Figure 4.5: Error Report Example

```
Input                          ::= atomic-expression | advanced-expression
atomic-expression              ::= 2-arg-atomic-expression | 3-arg-atomic-expression
advanced-expression            ::= 1-arg-advanced-expression | 2-arg-advanced-expression
2-arg-atomic-expression        ::= 2-arg-atomic-operator '(' variable ',' variable ')'
3-arg-atomic-expression        ::= 3-arg-atomic-operator '(' variable ',' variable ','
                                   'BGP'|'IP' ')'
1-arg-advanced-expression      ::= 1-arg-advanced-operator '(' variable ')'
2-arg-advanced-expression      ::= 2-arg-advanced-operator '(' variable ',' variable ')'
                                   | check-operator '(' variable ',' 'IP'|'submask' ')'
2-arg-atomic-operator          ::='match'|'matchv'
3-arg-atomic-operator          ::='nodeAdj'
1-arg-advanced-operator        ::='unique'
2-arg-advanced-operator        ::='join'
check-operator                 ::='check'
variable                       ::=sequence of characters
```

Figure 4.6: EBNF for Query Language

graph. The "protocol" argument to the "nodeAdj" command is used to choose between these two views.

| # | Operator | Detail | Return Value |
|---|---|---|---|
| 1 | match(x,y) | select data x from field y | The matched element sets |
| 2 | matchv(x,y) | select x whose value is y | Matched variables |
| 3 | join (x,y) | join two selected x and y results | The matched element sets |
| 4 | check(x,ip) | check whether x is IP format valid | Boolean value. If x is a valid IP return true; Else return false |
| 5 | check(x,prefix) | check whether x is a valid prefix | Boolean value. If x is a valid prefix return true; Else return false |
| 6 | unique(x) | check whether x is unique | Boolean value. If x is unique return true; Else return false |
| 7 | nodeAdj(x,y,protocol) | check whether x and y are adjacent nodes based on the protocol | Boolean value. If x and y are adjacent, return true; Else false |

Table 4.8: Predefined Operators

### Atomic Operators

An "atomic operator" is an operator that can be used as the operand to other operators. For example, if F(x,y) is defined as an atomic operator, F(x,y) can be used in operator S(F(x,y),z) as an operand. Atomic operators can also be used by themselves as operands. For given example F(F(x,y),z) is also a valid expression. An "atomic query" consists of an atomic operator and the operands required for the operator.

The atomic operators defined in the query language are:

- match(x,y)

  match(x,y) selects data x from field y.

  This operator locates the entries whose ancestor elements are y. For example, to search all router names in the network, the query is "match (hostname, host)". This query locates the name entries of all routers. The output of the query is the total number of matched hostname.

- matchv(x,y)

  matchv(x,y) selects data x whose value is y.

  This operator is the variant of match (x,y) operator. It locates the entries whose value matches the parameter y. For example, to search for the interface in the router with name S0-0/0/0, the query is "matchv(interface, S0-0/0/0)". The output of the query is the matched interface information in the global view.

- nodeAdj(x,y,protocol)

  nodeAdj(x,y,protocol) checks whether x and y are adjacent nodes based on IP and BGP protocols. The concept of "adjacent" is different for each protocol. For IP, the "adjacent nodes" means the nodes are in the same subnet. For BGP, the "adjacent nodes" means the nodes are configured as BGP neighbors. The operand x and y can be router names, or the IP addresses.

  For example, "nodeAdj(p1, p2, IP)" checks whether router p1 and p2 are connected for IP protocol while "nodeAdj(132.1.6.3, 142.1.5.2, BGP)" checks whether these two addresses are BGP neighbors. The output is a boolean value.

### Composition Operators

"Composition Operator" allows one level of composition for atomic operators. The operator itself cannot be used as the operand for other operators. For example, if

A(x,y) is defined as an atomic operator, C(X,Y) is defined as a composition operator, then C(A(x,y),z) is valid while C(C(A(x,y),z),z) is not valid since C(X,Y) operator cannot be used as the operand. The defined composition operators are:

- join(x,y)

  join(x,y) combines x and y together as the result.

  For example, to search all interface information and label switched path data, the query is "join(match(interface,interfaces),match(lsp, lsps))". Two "match" atomic operators are used as the operands to search the interface and LSP data separately. The "join" operator then combines the two results together. The output is a list of the matched information.

- check(x,ip/prefix)

  check(x,ip/prefix) checks whether x is a valid IP/prefix.

  For example, to check whether the IP address of an interfaces is correct, the query is "check(match(ipaddress, interface),ip)". The "match" operator is used to get the IP address of the interface. The "check" operator checks whether the matched IP addresses are valid. The output is a boolean value. If the prefix needs to be checked, the query is "check(match(ipaddress, interface),prefix)".

- unique(x)

  unique(x) checks whether the value of x is unique in the same configuration file.

  For example, to check whether the interface name is unique within one router, the query is "unique(match(name, interface))". The "match" operator is used to get the interface names for each router. The "unique" operator checks whether the name is unique. The output is a boolean value.

### 4.6.2 Rules and Query Language

The error checking rules can also be implemented by using these operators. Hence users can use the combination of the operators to create new verification rules. Table 4.9 gives more details about the predefined rules and corresponding operator combinations.

## 4.7 Tools for Implementation

XVT is implemented using Java and XML.

### 4.7.1 XML Document Parser

The parser used for XML documents is JDOM from http://www.jdom.org. The JDOM parser provides a Java-based solution for accessing, manipulating, and outputting XML data from Java code.

### 4.7.2 XSLT Engine

The XSLT (eXtensible Stylesheet Language Transformation) engine used in the application is Xalan-Java (version 2.3.1) from http://xml.apache.org. Xalan-Java is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (XPath). It is used as a module in the program.

### 4.7.3 XQL Engine

The XQL(XML Query Language) engine used in the application is GMD-IPSI XQL engine (Version 1.0.2) from http://xml.darmstadt.gmd.de/xql/. The GMD-IPSI XQL engine is a Java based storage and query application for large XML documents.

It provides a persistent implementation of W3C-DOM Document objects and a full implementation of the XQL language.

### 4.7.4 Visualization

A Java applet, graph.java, is used for the visualization of the router nodes and links in the Topology Extractor. The applet is a free open source tool from http:// www.gg.caltech.edu/ dhl/java/test/graph/graph.html that draw the nodes and links which are fundamental for the network topology representation. Furthermore the positions of nodes and links are not fixed in the display. The nodes can be dragged and the topology can be spread or folded.

| # | Rule | Operator Combination |
|---|---|---|
| 1 | "destination"field for label switched path cannot be null | match(destination, lsp)!=null |
| 2 | The VPN name must be unique in each router | unique(match(name, VPN)))=true |
| 3 | The path name in the named path set must be unique in each router | unique(match( name,path))=true |
| 4 | The interface name must be unique in each router | unique(match( name,interface))=true |
| 5 | The label switched path name should be unique in each router | unique(match( name,lsp))=true |
| 6 | There should be at least one label switched path | match(lsp,lsps)!=null |
| 7 | BGP neighbor cannot be itself | nodeAdj(x,x,bgp)=false |
| 8 | The BGP neighbors match each other | nodeAdj(x,y,bgp)=true |
| 9 | The label switched path should be referenced from the named path | match(match(pathname,lsp), match(name,path))=true |
| 10 | IP address of each interface is valid | check(match(ipaddress, interface),ip)=true |
| 11 | IP address of each router is valid | check(match( ip,host),ip)=true |
| 12 | IP address of each hop in named path is valid | check(match( address,path),ip)=true |
| 13 | IP address of BGP neighbor is valid | check(match( neighbor,bgp),ip)=true |
| 14 | Label switched path destination is valid | check(match( destination,lsp),ip)=true |
| 15 | IP address of neighbor of VPN BGP is valid | check(match(neighbor, match(bgp,VPN),ip)=true |
| 16 | Subnet Prefix of each interface is valid | check(match(prefix, interface),prefix)=true |
| 17 | The links defined in the explicit path should be configured | nodeAdj(hop1,hop2,IP) |

Table 4.9: Error Rules Expressed by Operators

# Chapter 5

# Test

## 5.1 Simulation of Environment

### 5.1.1 Network Topology

The network topology used for simulation is shown in Figure 5.1.

(1) There are four CE routers. CE1 and CE3 are in VPN1, while CE2 and CE3 are in VPN2.

(2) There are two PE routers, each one connected with two CE routers from different VPNs. One of the PE routers is configured as a Cisco router; the other one is configured as a Juniper router.

(3) There are ten P routers in the VPN core. Routers P1, P3, P4, P7 and P10 are configured as Cisco routers, while routers P2, P5, P6, P8 and P9 are configured as Juniper routers.

(4) There are two LSP tunnels in the topology. Both of them are configured as explicit-path LSPs.

We create the configuration files for all routers mentioned above and collect these files beforehand.

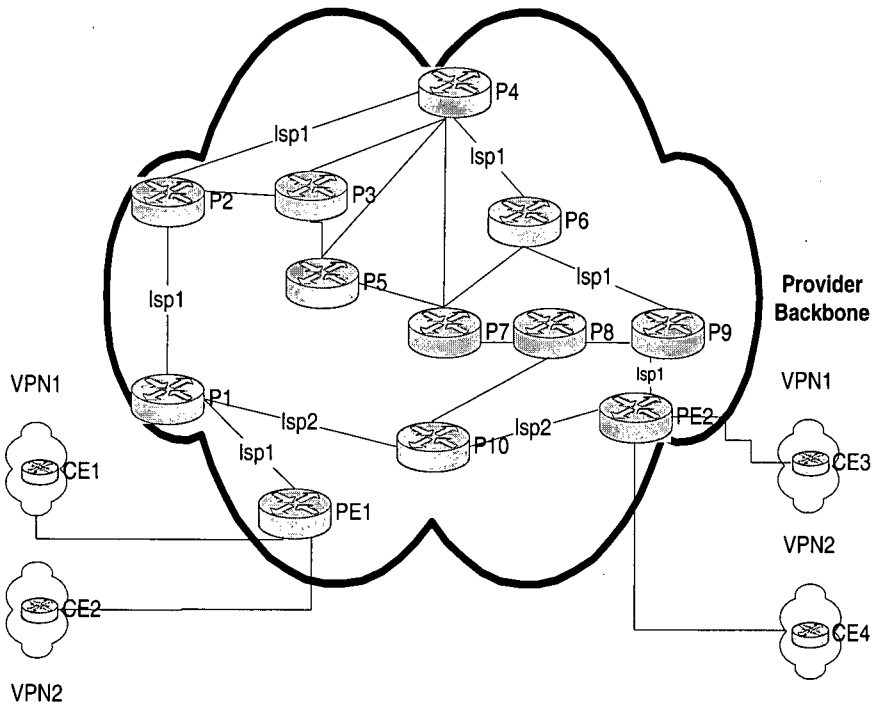



Figure 5.1: Network Topology

### 5.1.2 Router Configuration Files

All configuration files are collected before they are checked by XVT. XVT executes as an off-line tool and there is no data transfer between routers to be checked and the machine running XVT.

There are two types of router configuration files. One is for Cisco routers while another is for Juniper routers. Since there is no real router to set up the configuration files, the company's online router configuration manuals and documentation were used to create the configuration files.

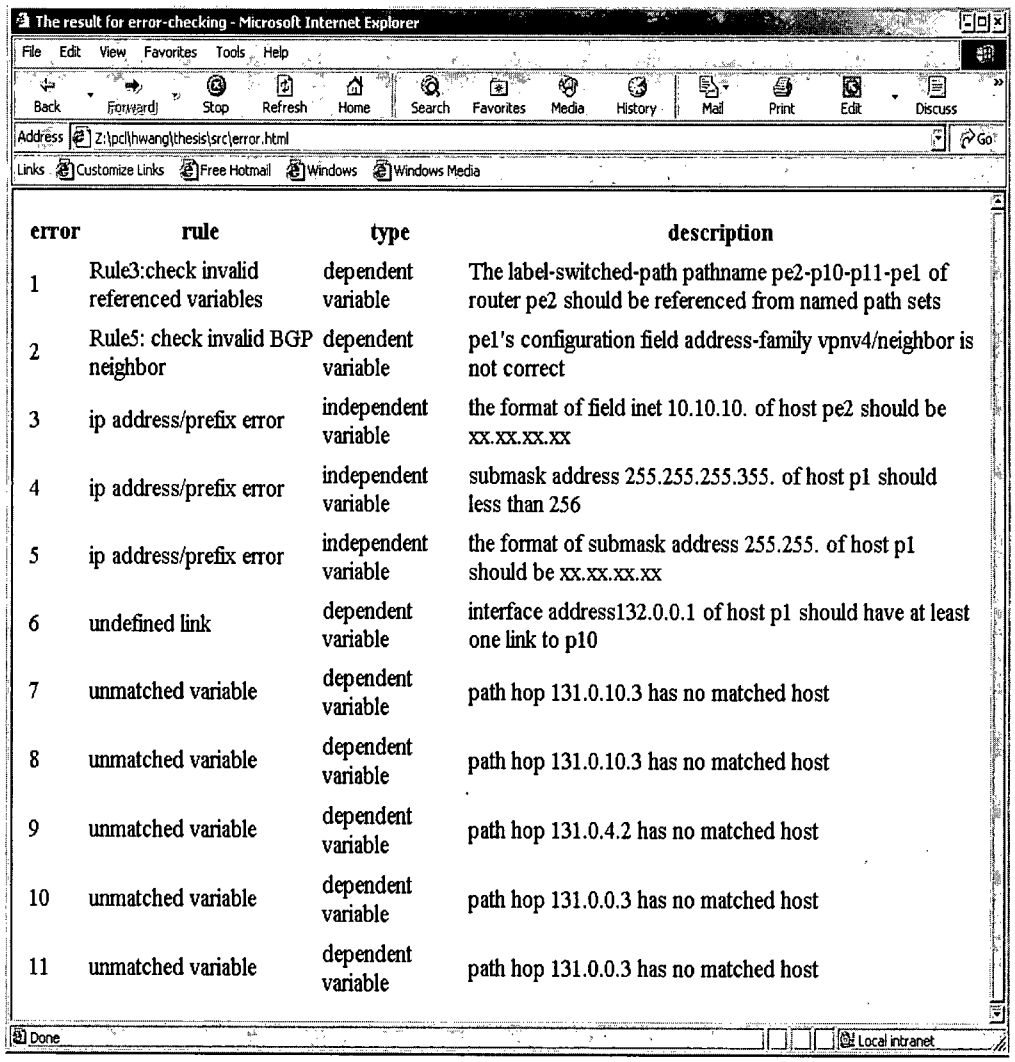
## 5.2 Error Report

The output error report is shown in Figure 5.2 for the defined topology. The error report consists of verification rules, error types and detailed description of the errors.

## 5.3 Results

The XVT tool is tested on a Linux machine with 128MB memory and Pentinum II 266 Mega Hertz(MHZ) CPU. The sizes of tested configuration files are from 400 bytes to 7548 bytes.

The performance results are shown in Figure 5.3. As the number of configuration files increases, the parsing time and checking time increases as well. The parsing time increases more rapidly than checking time for configuration files of different size. The reason is that parsing time is directly proportional to the number of configuration files while checking time is related to the time needed to query the global view. The global view is represented in an XML file and checking is performed by using XSLT so that the searching time is dependent on the processing speed of the XSLT engine. Although increasing the number of configuration files also increases the content of the global view, the XSLT engine reads the arbitrary XML elements without scanning the whole global view. As a result increasing the




| error | rule | type | description |
|---|---|---|---|
| 1 | Rule3:check invalid referenced variables | dependent variable | The label-switched-path pathname pe2-p10-p11-pe1 of router pe2 should be referenced from named path sets |
| 2 | Rule5: check invalid BGP neighbor | dependent variable | pe1's configuration field address-family vpnv4/neighbor is not correct |
| 3 | ip address/prefix error | independent variable | the format of field inet 10.10.10. of host pe2 should be xx.xx.xx.xx |
| 4 | ip address/prefix error | independent variable | submask address 255.255.255.355. of host p1 should less than 256 |
| 5 | ip address/prefix error | independent variable | the format of submask address 255.255. of host p1 should be xx.xx.xx.xx |
| 6 | undefined link | dependent variable | interface address132.0.0.1 of host p1 should have at least one link to p10 |
| 7 | unmatched variable | dependent variable | path hop 131.0.10.3 has no matched host |
| 8 | unmatched variable | dependent variable | path hop 131.0.10.3 has no matched host |
| 9 | unmatched variable | dependent variable | path hop 131.0.4.2 has no matched host |
| 10 | unmatched variable | dependent variable | path hop 131.0.0.3 has no matched host |
| 11 | unmatched variable | dependent variable | path hop 131.0.0.3 has no matched host |

Figure 5.2: Error Report Example

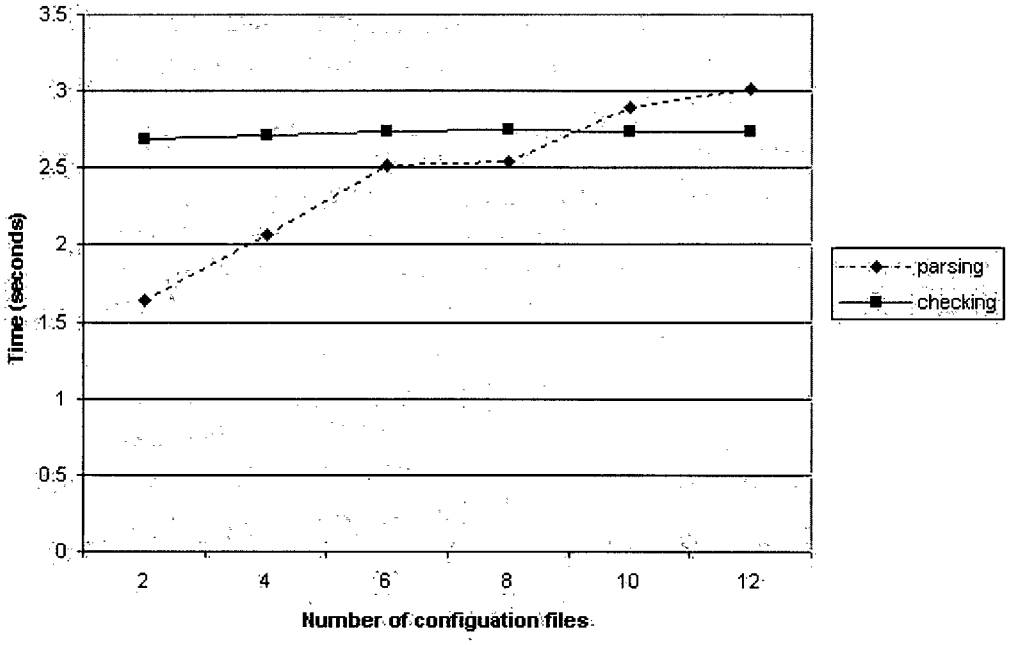



Figure 5.3: Performance Chart

number of configuration files does not greatly influence the checking time but does affect the parsing time.

# Chapter 6

# Conclusion

## 6.1 Conclusion

In order to support the various requirements of customers due to the growth of Internet services, network service provider companies have acquired network elements from various vendors. Multi-vendor configuration management, which is partly concerned with configuration verification, is complicated by the fact that each vendor uses a different configuration language.

The design of a general router verification tool over multi-vendor network is motivated by the idea of generalizing configuration data from different routers. XML is chosen as the general data repository. XML, which brings a mark-up language to data representation, is useful in data integration due to its ability to be machine accessible but in a human understandable way. The use of XML in our tool helps to express different document types because of its separation of data content from how it is represented.

We have implemented a router configuration verification tool, XVT, for multi-vendor networks. XVT currently can parse MPLS configuration files from Cisco and Juniper companies. XVT also can verify these configuration files against the predefined rules. Compared with existing tools such as OPNet's NetDoctor XVT provides predefined operators for users to define new verification rules. XML

is used in XVT to integrate various router configuration files into one uniform global view that is used later for verification purpose. The advantages of using XML are its flexibility in data representation, its ability to be easily transformed into HTML or plain text and the available XML-based application tools. Although XML is not suitable to represent the graph-viewed network topology, we have defined a set of graph-based query operators to overcome that disadvantage. Our experimental result showed that XVT can successfully verify the router configuration files from Cisco and Juniper against the predefined error checking rules.

## 6.2 Future work

The XVT tool described in this thesis is used offline. In the future we will enable XVT to collect the configuration files remotely from routers instead of using local files.

XVT only examines part of the router configuration files that is for MPLS protocol as well as related interface and global variables configuration. XVT only can parse and validate the configuration files from Cisco and Juniper. XVT could be extended to other protocols and vendors. A potential problem for the extension is that the design of a general parser for different types of router configuration files may be complicated because of various configuration languages used by the vendors. A possible solution is to provide a resource library which includes the router configuration language parsers for most of the router vendors. Each router configuration file will have a parser to interpret and map its data into the global view. If one specific kind of router configuration files is imported, the corresponding parser from the resource library is chosen to parse the newly added configuration files.

Our current verification operators are useful for verifying the consistency of configuration as well as links between two routers. Although most of our predefined verification rules can be represented by using these operators, it is still possible

that these operators are insufficient if XVT is to be extended to other protocols or other vendors. There may be new verification rules that cannot be represented by our operators. For example, there is no operator to test whether the area ID is contiguous or not for the Open Shortest Path First (OSPF) protocol.

# Bibliography


[1] Anja Feldmann, "IP network configuration for intradomain traffic engineering", *IEEE Network, Volume 15 Issue 5, Sept.-Oct.*, 2001.

[2] C.Alaettinoglu and C.Villamlizar, "Routing Policy Specification Language(RPSL)", RFC 2622, June 1999.

[3] L.Feldkhum,"Integrated Network Management Systems", *Proceeding of the first International Symposium Integrated Network Mangement*, 1989, Pages 279-301.

[4] OPNet Networks, Inc., *http://www.opnet.com/products/modules/netdoctor.html.*

[5] Lucent Networks, Inc., http://www.lucent.com/press/0399/990318.nsb.html.

[6] Dorado Software, Inc., http://www.doradosoftware.com/html/about/press/vpn-sc.shtml.

[7] William Stalling, "Network Management", *IEEE Computer Society Press, 1993*, Pages 1-11.

[8] Anja Feldmann, "Netdb: IP network configuration debugger/database", *AT&T lab, draft*, 1999.

[9] Juniper Networks, Inc., "Juniper JUNOS Internet Configuration Guide", *Juniper JUNOS Internet Software documentation Release 5.3*, May.01, 2002.

[10] Cisco NetWorks, Inc., "IOS Switching Services Configuration Guide".

[11] Chunk Semeria, "RFC2547bis: BGP/MPLS VPN Fundamentals", *White Paper, Juniper Networks, Inc.*, March, 2001.

[12] Cisco Networks, Inc., "MPLS Virtual Private Networks", *White Paper.*

[13] B.Neumair, Technical University of Munich, "Modelling resources for Integreated Performance Management", *Integrated Network Management III*, 1993, Pages 109-121.

[14] J.Case, K.McCloghrie, M.Rose and S.Waldbusser, "Management Information Base for SNMPv2", RFC 1450, April 1993.

[15] J.Case, K.McCloghrie, M.Rose and S.Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[16] J.Case, K.McCloghrie, M.Rose and S.Waldbusser, "Structure of Management Information for version 2 of the Simple Network Management protocols(SNMPv2)", RFC 1442, April 1993.

[17] International Organization for Standardization(IOS), "Specification of Abstract Synta Notation One (ASN.1)", International Standard 8824, December, 1987.

[18] Do-Hyeon Kim, You-Ze Cho, "An Efficient Integrated Network Management for heterogeneous LANs and WANs", *High Performance Computing in the Asia-Pacific Region. The Fourth International Conference/Exhibition*, vol.1, 2000, Pages 61-64.

[19] W3C organization, "W3C Recommendation XML 1.0", *http://www.w3.org/TR/ REC-xml* , Oct.6, 2000.

[20] David Lewis and Jens D.Mouritzsen, "The Role of XML in TMN Evolution", *Integrated Network Management Proceedings IEEE/IFIP International Symposium*, 2001.

[21] W3C organization, "W3C Recommendation XSLT 1.0", *http://www.w3.org/TR/ xslt*, Nov.16, 1999.

[22] W3C organization, "W3C work draft XML-QL", *http://www.w3.org/TR/ NOTE-xml-ql*, Aug.19, 1998.

[23] Xalan-Java, a XSLT processor, *http://xml.apache.org/xalan-j/*.

[24] W3C organization, "W3C work draft XQuery 1.0", *http://www.w3.org/TR/ 2001/WD-xquery-20011220/*, Dec.20, 2001.

[25] W3C organization, "W3C work draft XQL", *http://www.w3.org/TandS/QL/ QL98/pp/xql.html.*

[26] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, and Jennifer Rexford, "NetScope: Traffic Engineering for IP Networks", *IEEE Network Magazine, Special issue on Internet traffic engineering, AT&T lab,* 2000.

[27] Shane O'Donnell, "Network Management: Open source solutions to proprietarty problems", *SIGUCCS 2000*, Oct.29, 2000, Pages 214-216.

[28] OpenNMs organization, "OpenNMS brochure", *www.opennms.org/files/misc/ opennms5a.pdf.*

[29] James W.Hong, Michael A.Bauer and J.Michael Bennett, Departmeent of Computer Science, University of Western Ontario, "Integration of the Directory Service in the Network Management Framework", *Integrated Network Management III,* 1993, Pages 149-160.

[30] Yahoo Reuters Internet Report, "Cisco Router Market Share Edges Higher", *http://biz.yahoo.com/ri/020516/tech_cisco_study_2.html*, May 16, 2002.

[31] Network Working Group, "An Architecture for IP Address Allocation with CIDR", RFC1518, Sept.1993.